パソコンプログラマのためのクリエイティブマガジン

1991年6月1日発行(毎月1回18日) 第1巻第6号通巻6号 平成3年4月2日第三種郵便物認可

月刊プログラマーズ・ページ

Programmer's Page

MONTHLY MAGAZINE FOR PROGRAMMERS

1991 6 JUNE

定価980円



特集

研究「プログラミング統合環境」

快適で効率的な開発環境を考える

付録:5"2HDフロッピーディスク

本誌掲載プログラム／綴じ込み付録テキストファイル（統合開発環境メニュー一覧）／クイズ・ランダムスクリーン解読ツール／ファイル管理ユーティリティ FD Ver.2.30／高圧縮書庫管理プログラム LHA

SE
Shoeisha Co., Ltd.



美しさを追求する
パーソナルコンピュータはAXです。



■新登場・Turbo Vision

もはや、アプリケーションを書く際にユーザインターフェース等の細かい部分で苦労する必要はありません。新しいTurbo Pascal 6.0に付属する、オブジェクト指向のユーザインターフェース・クラスライブラリTurbo Visionを使えば、プロ開発者レベルのすばらしいユーザインターフェースを持つアプリケーションを、驚くほど簡単に記述することができます。これによって、位置・サイズを自由に変更可能なオーバーラップウィンドゥやプルダウンメニュー、ダイアログボックス、ボタン等を駆使したマウス操作のイベントドリブン・アプリケーションを短時間で作成できます。さらに電卓、カレンダー、エディタ、時計、ディレクトリブラウザ、フォームといったアプリケーションをサンプルとしてソースコード形式で提供。これらを部品としてアプリケーションに組み込むことができます。事実、Turbo Pascal 6.0の新・統合開発環境もTurbo Visionを使って開発されています。つまり、操作性では定評のある統合開発環境と同レベルのアプリケーションが簡単に作成できる訳です。

■大きく改善された統合開発環境

新しい統合環境は、オーバーラップマルチウィンドゥをサポートし、マウス操作も可能です。エディタは、1MBまでのファイルを同時に複数オープンできます。また、内蔵デバッガでは条件ブレークポイント機能やレジスタウィンドウも追加されています。

■組み込みアセンブラ

従来のインラインアセンブル機能に加え組み込みアセンブラが使用できます。ソースプログラムにニーモニックを直接記述できるため、PASCALプログラム中でも簡単にアセンブラ言語を利用できます。

■ハイレゾルーションモードに対応

統合環境のハイレゾ31行モードでの動作、BGIやTurbo Visionでのハイレゾモードをサポート。また、H98シリーズの拡張アトリビュートにも対応しています。

■Turbo Pascal 6.0 Professional

Professional版には、タフなバグを始末するTurbo Debugger 2.01、プログラムのボトルネックを取り除くTurbo Profiler 1.0、MASMコンパチブルな高速アセンブラTurbo Assembler 2.0に加えて、プロテクトメモリを利用して巨大アプリケーションがコンパイルできるコマンドラインコンパイラが付属します。

■Turbo Pascal 6.0の動作環境

機種：PC-9801シリーズ、PC-286/386シリーズ（ハイレゾモード対応、LTは除く）／メモリ：640KB以上／MS-DOS：バージョン2.11以上／マウス：バスマウス

■価格：Turbo Pascal 6.0………29,800円（税別）
Turbo Pascal 6.0 Professional…59,800円（税別）



TURBO PASCAL® 6.0

●製品に関するお問い合せは、総販売元㈱マイクロソフトウェア アソシエイツ TEL.03-3486-1411までお願いします。

株式会社ボーランドジャパン
〒107 東京都港区南青山7-8-1 小田急南青山ビル TEL.03-3486-1400㈹

<資料請求番号001>

8088から80486までの全命令に対応!!

コメント付きのアセンブラソースを作成する
世界で最も優れた逆アセンブラ

SOURCER

全てのMS-DOS
プログラマーに捧げる
限界を越えた
コード解析ツール

SOURCERはアセンブラやC, FORTRANなど、あらゆる言語によって作成された、様々なMS-DOS実行形式ファイルから、処理内容に対応した適切なコメントを伴った分かりやすいアセンブラ・ソースリストを作成する、リバース・エンジニアリングに不可欠なコード解析ツールです。

解析の正確さ、出力ソース・コードの分かりやすさ、さらに解析可能な命令セットの範囲の広さ、操作性の良さなど、SOURCERはあらゆる面で他に類を見ない優れた逆アセンブラです。

こんな時に威力を発揮します

■ドキュメントのないソースコードの解析を行いたい
■ソースが入手不可能なプログラム(市販の各種ソフトウエア)に対して修正を行いたい
■プログラム(市販の各種ソフトウエア等)が実際どのように動作しているかを知りたい
■ソース・コードを紛失してしまった
■「ウイルス」を含んでいる恐れのあるプログラムをソースコード・レベルで検査したい

SOURCERの特長

●以下の命令セット実行プログラムの解析逆アセンブルが可能です。

8086/87/8088　80186/188　80286/287
80386/387　80486　V20/30

●以下の種類の実行形式プログラムに対して、ユーザが指定する任意のアドレス範囲に対して逆アセンブルが可能です。
▶COMファイル　▶EXEファイル　▶デバイス・ドライバ
▶オーバーレイ・ファイル　▶SYSやBINなど、その他のバイナリ・ファイル
▶メモリ上に既にロケーションされている(ROM領域も含めて)プログラム

●複雑に入り混じるマルチ・セグメント構成に対して、最大250個までのセグメントの解析が可能です。

●各種アセンブラ処理系に特定対応した逆アセンブラ・ソース・リストの作成もできます。

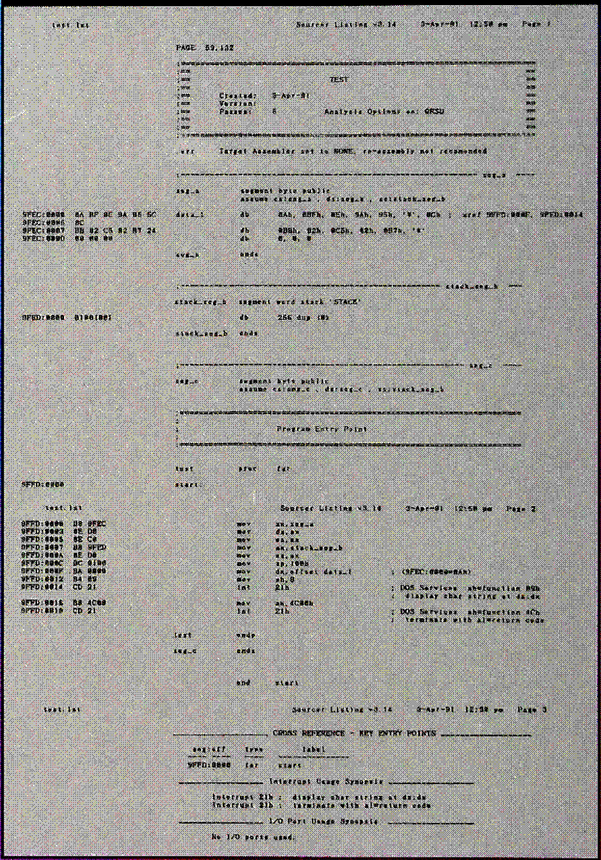
MASM-4.0　MASM-5.0　MASM-5.1
TASM-1.0　TASM-2.0　OPTASM　その他

●解析結果の逆アセンブル・リストには、使用されているファンクション・コールの機能と各レジスタの意味、アクセスしているデータ領域に関する細かい情報、I/Oポートの操作状況、複雑なコードの処理内容などについての、詳細なドキュメントに類するコメントが付加されます。

●解析オプションを指定すると、解析方法を、ユーザ側で細かく設定できます。さらに、定義ファイルを使用することによって、逆アセンブル・ソース・リストのカスタマイズも行えます。

●最大9パスまでの綿密な解析逆アセンブリ処理を行います。

解析逆アセンブルリスト出力サンプル



価格:¥38,000

【動作機種】▶PC-9800シリーズ ▶FMRシリーズ ▶J3100シリーズ ▶Panacom Mシリーズ ▶PS/55シリーズ ▶AX
【動作環境】MS-DOS Ver. 2/3/4
必要メモリ:512KB以上
【供給メディア】5インチまたは3.5インチ フロッピー・ディスク

完全日本語マニュアルを装備

【お求め方法】当社製品は、パソコンソフト取扱い店でお求めいただけますが、弊社に直接ご注文の際は、The Programmer's INDEX (TEL:03-3293-3887)までお問い合わせ下さい。※各製品の表示価格には、消費税は含まれておりません。

株式会社 LIFEBOAT
〒101 東京都千代田区神田錦町3-6
PHONE:03-3293-4711代 FAX:03-3293-4710

Brief

Full Screen Editor 日本語版

全米のプログラマが認めた実力！
アメリカでは「ブリーフ」が
プログラミング用エディターとして
最も多く利用されています。

C言語で組めるマクロ！

Version UP!

V.3.0 いよいよ登場。
プログラミングエディター新世紀。



よりパワフルで、よりエレガントな
プログラミングのために……。

●C言語で組めるマクロ 新機能
使い慣れているC言語でマクロが組めますから、習得が簡単でカスタマイズも思いのままです。

●マクロのスクリーン・デバッガ 新機能
ソースコードでデバックできます。もちろんブレークポイントやウォッチ・ドッグもセットできます。

●テンプレート機能〈構文の自動生成〉 新機能
C、Ada、Basic、COBOL、FORTRAN、MODULA2、Pascalで構文を自動生成します。

●300回ものアンドゥ
各ファイルごとに300までの操作を元にもどすことができますので、やり直しの面倒を気にせず、安心して編集できます。

●安全編集
Briefは、一定時間キーを操作しないこと最新の編集内容をファイルに自動保存します。だから編集中に電源が切れても安心です。

●その他の主な新機能
キー操作のファイルへのセーブとリプレイ、処理可能行数を拡張、リジューム機能、Ctrlキー・アサイン（オプション）、メニュー（オプション）etc.

アメリカでも日本でも高い評価
●プログラミング向きエディターを探している人は、もう探す必要はない。(Jerry Pournelle BYTE)
●ソフト開発には強力なツール、300回のアンドゥは大きな特徴。(日経バイト)
●既存のエディターに不満を覚えたら、これしかない。(Oh! FM)
●まるで夢のようなアンドゥだ　おもわず間違うのを期待してしまいそうだ。(The C Users Journal)

■主な機能
◆55ウィンドウ◆無限のファイル◆自動コンパイル◆自動エラー・ジャンプ◆自動インデント◆自動テンプレート◆バッファ書出　切換　削除　リスト表示◆ファイル名の指定　表示　変更◆バックアップ保存◆特殊文字入力◆2行を1行に連結◆ファイル単位の挿入◆増分検索◆各種マーク　始・終点の入換◆ブロックのカット　コピー　ペースト　印刷　保存　左右へのシフト◆対応する括弧へ移動◆C言語のコメント化　非コメント化◆DOSコマンドの実行◆キー操作記憶　実行◆入力ヒストリー◆英文テキスト整形◆タブのスペース挿入◆拡張子毎に異なった設定ができるタブ◆検索置換パターンの画面からの入力◆英字の大小文字変換　その他

定価 48,000円
（消費税は含まず）

■5機種対応パッケージ
（東芝、NEC、AX協議会、IBM、富士通）
●5.2DD、3.5 2DD デュアルメディア●MS-DOS3.1以降に対応●英語DOSでは使用できませんが、英語版BRIEFもセット販売いたしております。●お急ぎの方、上記以外のメディア希望の方、英語BRIEFのセット販売希望の方はユーザー・サポート係までご連絡下さい。

日本システム株式会社

●お問合せは　東京都立川市曙町1-18-2一清ビル立川事業所Briefユーザー・サポート係
TEL0425-23-4469　FAX0425-27-6127

〈資料請求番号005〉

Programmer's Page

MONTHLY MAGAZINE FOR PROGRAMMERS

特集
研究「プログラミング統合環境」
快適で効率的な開発環境を考える

表紙イラスト：日暮修一
表紙AD：佐々木幹夫(SE)

付録
5"2HD フロッピーディスク

綴じ込み付録
統合開発環境メニュー一覧

1991 JUNE No.6

SE 翔泳社が贈るプログラミングブックス



アセンブラプログラミング撤底解説宣言
アセンブラプログラマー必携の書登場！

マクロアセンブラ プログラマーズテキスト

吉川敏則 著■定価2,600円（税込）

PC-9801シリーズの代表的なCPUであるインテル社のi8086系列（i80286，i80386）および標準オペレーティングシステムのMS-DOSを活用するためのマクロアセンブラ（MASM）を使ったプログラミング書。最低限度必要なハードとソフトの知識から、データ演算の仕組みを踏まえた上で、アセンブラプログラミングの基本を丁寧に解説します。また後半では、実践的なアセンブラテクニックをMS-DOSファンクション・リクエストの活用法から学び、Code Viewによるデバッグ方法で締めくくります。



C言語入門書異説

Cの話

安良城勝也 著■定価1,500円（税込）

赤ん坊が言葉を学ぶときは、「はじめに文法ありき」ではない。言葉のシャワー、情景のシャワーの中でセンスが磨かれ、突如として天啓が訪れる。プログラミング言語を学ぶにも、まずセンスの醸成が必要だ。本書はC言語を学ぶ前に読む本。ソフトウェアとは何か、Cのプログラムとはどんなものか…軽いタッチで一気に読める、頭の準備体操。

SE 株式会社 翔泳社
東京都千代田区平河町2-4-14 平河町KSビル 〒102
PHONE 03-3263-0447 FAX 03-3263-4526

〈資料請求番号003〉

特集

研究「プログラミング統合環境」

快適で効率的な開発環境を考える

かつて，プログラミング開発環境が論じられたとき，まず第一に求められたのは生産性の向上，すなわちいかに効率よく開発するかということであった．その効率とは，膨大なソフトウェアを限られた期間に開発するという意味であり，時間を有効に使って個人の創造性を高めるものではなかった．ソフトウェアの作成が，ほんとうの意味で創造的な作業であるためには，プログラミングはもっと快適であるべきだ．それには，強力なプログラミング言語とともに，自分の手足として十分な役割をになってくれるプログラミング環境が不可欠である．パーソナルコンピュータの普及とともに個人のための開発環境が整備されてきている．さて，最新のプログラミング環境は，どこまでプログラマのニーズを満たしてくれるのだろうか？

特集 研究「プログラミング統合環境」

快適で効率的な開発環境を考える

プログラム作成支援環境の動向

用途により言語と支援環境の形態は異なる

鈴木則久

プログラミング統合環境を語るうえで，まず検討しておかなければならないのはプログラミングの「環境」とは何かということだ．それはプログラムの作成を「支援する」ためにソフトウェア，ハードウェアにどんな要件が求められてきたかということにほかならない．しかし，使用するコンピュータとOSの種類，コンピュータの利用形態，そこで使われるプログラミング言語，さらにプログラマの資質などによって，求められる要件も大きく違ってくる．ここでは，プログラミング環境を分析し，その発展の経過をたどることによって，プログラミングにおいて，なぜ「統合開発環境」が求められるかを考えてみよう．そして，それがどのような分化をとげていったか，またどこにいこうとしているかを探ってみたい．

プログラミングの形態と作成支援環境

ソフトウェア作成を効率化するための流れ

◆個人の能力に依存した研究用プログラミング

プログラムの作成能力は人によってずいぶん違います．研究者のように，プログラムに対するクリエイティビティの高い人が，自分のアイデアをソフトウェアにするということを考えてみましょう．ソフトウェアを作るには，まずプログラムを書いて，次にこれをコンパイルします．虫があったら取って，再コンパイルする．そこで，虫を取る時間や再コンパイルにかかる時間が短くなれば，クリエイティブな人は，もっとたくさんのすばらしいプログラムが作れるだろうという考え方があります．

つまり，ひとりの天才的なプログラマのプロダクティビティを高めればよいという考え方です．これは，プログラム作成支援環境の大きな目的のひとつでした．このような考え方のもとでは，できあがったプログラムの実行速度や効率についてはそれほど問題にされませんでした．Smalltalkにしても，最終的な製品を作るのではなく，プロトタイプの作成が中心だったのです．

このようなプログラミングの形態での作成支援環境として，非常に重要な機能であったのもののひとつはソースコード上でのデバッギングです．プログラムの実行中にどこで脱線したのかが，ソースコード中でわかる，プログラムを任意の場所で止めて，変数の中身を知ることができる，ということが重視されました．

2番めに重要な点として，ポインタの取り扱いがあります．虫のなかで，もっともわかりにくいものは，プログラムの進行とは無関係のデータ構造やプログラムが壊れてしまうというものです．こういう場合，その原因はたいていポインタにあります．ポインタが予期しない場所を指していて，そこへ誤って値を代入してしまうから起こります．このようなポインタをダングリングポインタといいます．ダングリングポインタが絶対起きないようにするには，動的な記憶の割り当て，ガーベッジコレクタが必要になってきます．

3番目はプログラムをオブジェクトという形で再利用するということです．

Smalltalkでは，この3つを実現することが目標とされていました．

◆グループでのシステムプログラミング

プログラミングのもうひとつの流れはUNIXとCです．個人でプログラムを作るのではなく，グループでのプログラミングによって，OSや通信ネットワーク用のソフトなどのシステムプログラムを作成するというプログラミングの形態です．これは，Smalltalkのように，研究用のプログラムを作成するのではなく，実際に販売するプログラムを作るのですから，開発の能率をあげるということが重要視されました．そのため，ソースコントロールシステム（STCS）のようなものが作られたのです．C言語のデバッガもかなり充



鈴木則久
日本アイ・ビー・エム（株）
東京基礎研究所所長

［略歴］
1946年生
東京大学大学院工学系電子工学修士課程終了後，スタンフォード大学博士，カーネギー・メロン大学助教授，ゼロックス　パロ・アルト研究所研究員，東京大学工学部計数工学科助教授，IBM T.J.ワトソン研究所研究員，日本アイ・ビー・エム（株）サイエンス・インスティチュート所長を経て1986年9月より現職

［著書］
「Automatic Verification of Programs with Complex Data Structures」（Garland Publishing 1980）「プログラミング言語とVLSI」（共著，岩波書店 1985）「オブジェクト指向」（編　共立出版　1985）「Smalltalk」（産業図書 1986）「研究者のテイストー1990年代のコンピュータ」（共立出版　1990）

統合開発環境用語集
IDE　ボーランド社のコンパイラ統合開発環境の名称．Integrated Development Environmentの略．

実したものが作られました.

その後，システム作りの環境をもう少しよくしていきたい，しかし，Smalltalkのようなハイレベルな言語では実際に多くのものを作るのはたいへんだということもあって，システム作りに適した開発言語を新たに作るという動きが出てきました．代表的なものとしてはXEROXのCedar，DECのModula-3，そしてIBMの基礎研究所にはCOBというプロジェクトがあります.

MesaやModula-2，Cといったシステム開発言語でも，効率のいいシステムプログラムは書けるのですが，これらの言語ではデバッグするのがたいへんである，プログラムの再利用がしにくい，といった欠点があります．このような問題を解決するために，言語の仕様そのものをSmalltalkのようなものに近づけて，ガーベッジコレクションやソースコードデバッギングの機能を取り入れようというのが新しい開発言語のねらいです.

これらの言語でシステムを作ろうという動きもありますが，それだけでOS全部を作れるかというと，能率はいま一歩であるという気がします．CでOSを作ることさえ，最初はずいぶんと抵抗があったくらいですから，ましてCよりももっと高級な言語ではなおのこと抵抗があります．今後コンピュータは，どんどん能力があがっていきますので，最後の最後まで能率を追究していかなくても，ある程度の能力がある言語であればかまわないともいえますが，これらの新しい言語はひとつの大きな潮流ではあります.

◆カストマープログラミング

こういったシステムプログラミングとは別に，カストマーが自分で使うためにプログラムを作成するという場合があります．いまでは，最初から全部を書かなくてもLotus 1-2-3のようなスプレッドシートのプログラムも売っていますが，それだけではすまないカストマーもいます．そういった，カストマーのための簡易言語というのができています.

最近ではマルチメディアということで，パソコン上でいろいろなグラフィックスを扱いながら，ソフトウェアを作っていかなければならないということもあるでしょう．このようなプログラミングにおいては，OSなどのシステムプログラミングの場合ほど，能率ということに注意は払われません.

ウィンドウ作成用の言語としての

新しいプログラミング言語

●Cedar

XEROXパロ・アルト研究所（PARC）で作成されたプログラミング環境．約4MIPSの処理速度をもつワークステーションDorado上で動作することを前提として，1978年から設計が開始された．当時，PARCではMesa，Smalltalk，InterLispという3つのプログラミング言語が使われていたが，Cedarは Mesaをベースにして設計された.

Cederプロジェクトの目的は，いかにプログラマの生産性を向上させるかということにあった．Cedarの設計にあたって重視されたのは，①いらなくなったオブジェクトを自動的に回収するガーベッジコレクション機能，②コンパイル時にエラーをチェックできる型システム，③主記憶とファイルの間でのオブジェクトやページのスワッピングを自動的に行う記憶管理，④言語としての抽象機構，⑤小さなプログラムの変更はすぐできて，実行できること，⑥情報隠ぺい，防御機構をもつこと，⑦データベース・アクセス機構を言語に取り込むこと，⑧実行時に型をもてる自由なデータ構造，⑨モジュールの依存関係に基づいて，矛盾なく最小限のコンパイルを行うコンパイラ，⑩バージョン管理のためのツール，などの機能であった.

Cedarシステムは1982年のはじめには利用可能となり，1984年中旬にはコンパイラやランタイム・ルーチンのサポートから電子メールシステムまでに及んだ．Cedarのインターフェースは，タイル型のウィンドウシステム，アイコン表示を採用し，「ホワイトボード」という文字と単純な図形を収めたページの集まりを処理する機能をもっていた．ワークステーション用としては1984年当時，最先端のプログラミング環境を提供するものだったといわれる.

●Modula-3

米Digital Equipment社（DEC）のSystems Research Center（SRC）とOlivetti Research Centerの研究者によって開発されたオブジェクト指向のプログラミング言語．N.Wirth博士が設計したPascalとこれにモジュール化の概念をつけ加えたModula-2，XEROXのPARCで開発されたMesa，Cedarの2系列の流れを受け継いでおり，基本的な言語仕様はPascal，Modula-2，Adaなどに似る．Modula-3の特徴は，Modula-2を一段とすすめたモジュール化，オブジェクト指向プログラミング，およ

統合開発環境用語集
アドバイザ　Microsoft Advisorは，Microsoftの言語ソフトで実行されるオンラインヘルプシステムの名称.

Smalltalkは，OS/2のウィンドウまわりの開発用として使われるようになりました．HyperCard用の言語HyperTalkなどもこれにあたります．

◆ソフトウェア工場

日本のソフトウェア会社，とくに大企業の子会社的なソフトウェア会社などには「ソフトウェア工場」という考え方があります．仕様の比較的かっちりと決まったもの，能率よく，かつ高い信頼性のあるものを作ろうという考えです．FORTRANのようにすでに言語仕様が定まった言語を使って，ソフトウェアエンジニアリングワークベンチで開発する．つまり，プログラマをよく管理して，ハードウェアを工場で生産するのと同じようにソフトウェアを作っていくという発想です．

以上が，現在のプログラミング作成支援環境の趨勢ではないでしょうか．

ガーベッジコレクション

ヒープ領域にメモリを確保する作業を繰り返すと，すぐ空き領域がなくなってしまう．そこで，適当なタイミングですでに利用されなくなったメモリ領域を回収して，再配置し，もう一度利用できるようにするという作業が行われる．この処理をガーベッジコレクション，またはごみ集めと呼ぶ．

C言語では，メモリの確保(malloc)や解放(free)はプログラマに委ねられているが，Lisp，Prolog，Smalltalkなどの言語ではガーベッジコレクションが行われるため，プログラマはこういった記憶管理にわずらわされることはない．

びスレッドモデルによる並行プログラミングである．

Modula-3のモジュールは，モジュールの外からこれをアクセスする場合にアクセス可能な型，オブジェクト，手続きなどのインターフェースを記述する部分と，そのモジュールの宣言や手続きを記述する部分の2つで構成されている．モジュール化によって，情報の隠ぺい，言語の抽象化が強力にすすめられている．

オブジェクト指向プログラミングの機能としては，同じクラスに所属する複数のオブジェクトを生成時に別のメソッドをもたせることも可能である．また，自動的にガーベッジコレクションを行う機能をもつため，コンストラクタ／デストラクタがない．

同一プログラムの同一アドレス空間で，複数のスレッドが並行に動作することができる．

SRC版は，ソースコードの形で配布されていて，すでに数多くのUNIXワークステーションに移植されている．

●COB

日本IBM東京基礎研究所で開発されたCをベースとするオブジェクト指向言語．設計にあたっては，MesaやCedarの仕様や思想の影響を大きく受けており，Cと互換性のあるコンパイラ型のオブジェクト指向言語であること，型システムを強化し，実行時の型チェック，配列のインデックスチェックなどを行うこと，オブジェクトの記憶管理はガーベッジコレクションによってシステムが一元管理すること，再コンパイルに要する時間をできるだけ短くすること，プログラムが完成した時点で強力な最適化を行い，実行効率を高めることなどが考慮された．

Cをベースとしたオブジェクト指向言語としては，C++が有名だが，C++ができあがったプログラムの実行効率を高めることを重視しているのに対し，COBではこれと同じ程度にプログラムの開発効率を高めることが重視されている．

COBの処理系はCOBプログラムを読み込んで，Cのコードを生成するという形で実現されており，すでにこの処理系はIBMの各種ワークステーション，パソコン上で動作している．

プログラミング言語と作成支援環境

Cの限界を超えるには仕様の変更が必要

プログラミングの作成支援環境というのは，使われる言語や用途によって大きく異なります．たとえば，Smalltalkの環境があれば，必ずアイデアをプログラムに生かすことができるのかというと，それは用途によります．ユーザーインターフェースまわりのプロトタイプなどを作成するには，SmalltalkはCよりもはるかに効率的ですが，これでネットワーク通信のソフトを作ることはできません．

Cの開発環境については，UNIXなりTurbo Cのようにパーソナルコンピュータ上で動作するものがありますし，それとは別にFORTRANならFORTRANで，科学計算という用途に適した作成支援環境があります．

たとえば，スーパーコンピュータがありますね．スーパーコンピュータには，ベクトル型のマシンと並列型のマシンがあります．ふつうにプログラムを書いてもってきただけでは，どちらのマシンでもそんなに速くは走りません．プログラムのベクトル化率とか，並列化率とかいうのが非常に性能に響いてくるのです．そうすると，プログラムのベクトル化の度合い，並列化の度合いを見られるようにビジュアライゼーションしなくてはなりません．FORTRANでは，こういったビジュアライゼーションが非常に重要なプログラム作成支援環境だということになってきます．

パーソナルなプログラミングの作成支援環境というのは，現在のところもういきつくところまできて，ある程度完成した形になってきています．そこで，現在では，先ほどもお話しましたが，プログラミング言語の仕様そのものを改良しようという方向にあります．C言語では，メモリの変なところがつぶされてしまったり，配列がきれいに取り扱われていないためにプログラムが暴走してしまうといったことはどうしても起こりえます．C言語そのままではどうしても限界があるのです．まずは言語仕様をきれいにして，その言語仕様を基礎としてプログラミング環境を整えていくということになるでしょう．(談)

コンピュータとプログラミング環境の過去・現在・未来を知る　編集部

●「ワークステーション原典」
ACMプレス編／村井純監訳／浜田俊夫訳／アスキー出版局／8,000円
原題は「A History of Personal Workstations」．1986年1月，パロ・アルトにおいてACM(Association for Computing Machinery）が企画した同名の会議における講演，論文，参考論文などを収録した本．現代の対話型コンピュータ環境設計分野のパイオニアであり，マウスの特許を持つダグ・エンゲルバート氏や（「知識増大ワークショップ」），ラップトップサイズのパーソナルコンピュータ「ダイナブック」を考案したアラン・ケイ氏（ダイナブック－その過去，現在，未来」)，Altoの設計者であるチャールズ・P・タッカー氏（「Altoとイーサネットハードウェア」)，Altoをはじめ数種のオペーレーティングシステムを設計したことで知られるバトラー・w・ランプソン氏（「パーソナル分散コンピューティング－Altoとイーサネットのソフトウェア」）の講演などが収められている．内容が学術的にも高度で，しかも分量も多い（約600ページ）が，パーソナルワークステーションの発達を知る上では欠かせない文献といえる．

●「情報処理　Vol.30 No.4　大特集：新しいプログラミング環境」
(社)情報処理学会／オーム社／2,000円
情報処理学会の会報である「情報処理」の，プログラミング環境に関する特集号．ソフトウェアの開発環境についての当時（1989年4月発行）の最新動向の紹介を目的としている．
1．プログラミング言語と環境，2．プログラミング・パラダイムと環境，3．プログラム設計環境のツール，4．新しい技術の応用，5．事例，6．究極のプログラミングの6部から構成されている．

●「コンピュータ言語進化論」
H.レヴァイン ＆ H.ラインゴールド著／椋田直子訳／アスキー出版局／1,960円
原題は「The Cognitive Connection」．本誌3月号特集でもとりあげたが，プログラミング言語を考えるうえでは，非常に興味深い1冊．言語は思考を表現するための手段としてとらえることができる．方言まで含めれば数百は存在するプログラミング言語にも，それぞれがある目的を持って作られている．おのおののプログラミング言語の成立の背景や，その目的，また今後の発展などを知るのに格好の書である．

統合開発環境用語集
アンドゥ　直前の複写や削除，移動など編集操作を取り消す機能

言語製品の流れとプログラミング環境の発展

編集部

プログラミングの環境とは

パーソナルな開発環境の前提

◆コンピュータの形態と開発環境

コンピュータといえば大型計算機，それをバッチジョブで利用する形態が中心であったころには，プログラム開発作業とはコーディングシートに書き込むことであり，使用するツールは鉛筆とフローチャートであった．タイムシェアリングシステムによって，ようやくプログラムの入力，編集，コンパイル，リンク，デバッグなどをすべて端末上で行えるようになる．だが，その端末もつねに他人と共有しなければならなかった．プログラマがそれぞれCPUをもったマシンを占有することが可能となって，はじめて現在のようにパーソナルなプログラミング環境の基礎が築かれたといえるだろう．

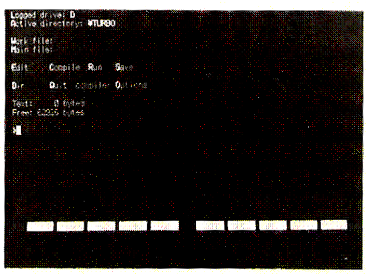
◆プログラミング言語と開発環境

プログラム開発環境を語るうえでもうひとつ忘れてはならないのが，プログラミング言語との関わりである．1979年に発売されて，日本のパソコンの歴史を開いたNECのPC-8001にはBASICがROMで内蔵されていた．その後，発売された各社のパソコンの多くがこのようなBASICマシンであったため，日本におけるパソコンのプログラム開発ではBASICが標準言語となった．これらのBASICマシンにおける開発環境は，現在のMS-DOSやOS/2上での開発環境よりもはるかに貧弱なものであった．当時使われていたBASIC言語の仕様自体も，大きく複雑なプログラムを作成するのに適していなかった．逆にいえば，それほど複雑な長いプログラムを作ることもできなかったので，開発のための環境のよしあしが問われることなどもなかった．

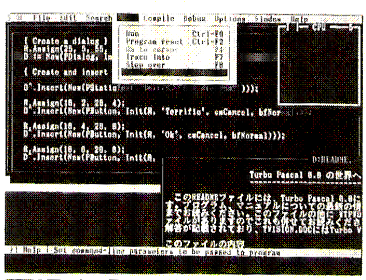
初期のプログラミング統合環境

Turbo Pascalのインパクト

パソコン上にCP/MやMS-DOSなどOSと呼べるものができると，そのうえで動作するプログラミング言語が登場してくる．1981年アメリカではすでに16ビットパソコン，IBM PCが発売され，当初からOSとしてPC-DOS（MS-DOS）を採用し，BASICとPascalに対応していた．



画面1 Turbo Pascal Ver.3.0



画面2 Turbo Pascal Ver.6.0

1983年，PC-DOSが2.0にバージョンアップしたのと同じ年に，Borland International社（以下Borland社）からTurbo Pascal 1.0が発売される．これはCP/M，CP/M-86，MS-DOSなどの各種OSに対応したPascalコンパイラで，コンパイル速度が高速なこと，低価格であったこと，またアメリカではPascal言語が教育用として早くから大学などで採用されていたこともあって爆発的に売れた．

実際のTurbo Pascalは，画面1のようなものであった．プログラムのソースファイル名を入力し，メニューからEdit（内蔵エディタを使ったプログラムの編集），Compile（コンパイル），Run（実行），Save（ファイルの保存），Dir（ディレクトリリストの表示），Quit(終了)，compiler Options（コンパイラのオプション変更)のいずれかを選択するようになっている．メニューだけを見ると，たしかに現在「統合環境」をうたっている多くのコンパイラとよく似た構成になっているが，メニュー構成そのものは単純で，インターフェースもかなり違っている．画面2に同じTurbo Pascal Ver.6.0をあげたが，双方を比較してみるとその違いがいっそうはっきりわかるだろう．

日本でも84年にはTurbo Pascal Ver.2.0が発売された．24,800円という価格は当時驚異的な安さであった．

エディタ付きの環境はBASICインタプリタに慣れたユーザーにも，受け入れやすいものだったのか，MS-DOSの普及につれて，Turbo Pascalは好意的に受け入れられていくこととなる．

Microsoftの参入

QuickBASIC

Borland社ではPascalでの成功をもとに，CやBASICなどほかの言語製品をも出していくと発表した．これに危機感をいだいたのが，Microsoftである．Microsoft社はもともと現会長のビル・ゲイツ氏らが開発したBASICを販売するために設立されたという経緯もあって，ことBASICに関しては強い自負をもっていた．そのMicrosoftがTurbo Pascalばりの統合環境付き処理系として世に送り出したのが，QuickBASICだ．同社では統合化された環境をもったBASICが，いかに使いやすく便利なものかということを積極的にアピールした．その結果，QuickBASICはユーザーに浸透し，言語処理系プログラムのひとつの形として，エディタ付きの環境という形態が確立する．ただ，残念なことに，このQuickBASICの初期のバージョンは日本語化されなかった．後にQuickCと前後してQuickBASIC4.2が日本語化される．このときは，なじみのあるBASICの製品であること，インタプリタとコンパイラの両方が使えることなどからちょっとしたブームにもなった．

PascalとBASICという違いはあれ，BorlandとMicrosoft 2社の競合関係が生まれ，その関係は後のC処理系においても引き継がれるのである．

C言語の普及と開発環境の発展

MS-DOSのプログラム開発環境

AT&Tベル研究所のRitchieによって作られたC言語は，UNIXとともにさまざまなマシンに移植される．アセンブラにくらべればはるかに読みやすく，分割コンパイルが可能で，しかもシステムレベルのプログラムまで記述できることから，また，パソコンで動くアプリケーションプログラムの開発需要が著しく増加したことから，C言語はパソコンの世界でも急速に標準的な開発言語へと昇格していった．

Microsoft社は，はじめLattice CのOEM供給を受けてCコンパイラを販売していたが，自社の開発言語としてもCを採用することを決め，1985年にはついに自社開発のMS-C Ver.3.0を発売した．8086アーキテクチャを有効に利用し，リンカやシンボリックデバッガを備え，アセンブラとのリンクも可能なMS-Cは，あっというまにパソコンC言語処理系のスタンダードとなる．翌年バージョンアップされてVer.4.0となったMS-Cには，ソースレベルデバッガCodeViewが付属していたため，MS-Cの人気はさらに高まった．

プログラミングの環境という点でみると，MS-Cには標準でコンパイラ，リンカ，デバッガなどのツールが含まれていたが，それぞれのツールは独自に機能するものだった．プログラマは，別途に専用のスクリーンエディタを購入して，それを使ってプログラムを作成，編集し，保存する．次にコンパイラを起動して，ソースファイルをコンパイルし，リンカを起動してリンクし，エラーがなくなったら実行する．そして，実行時にエラーが発生すれば，デバッガを使ってデバッグする．MAKEのように，ファイルの依存関係を記しておけば，あとは順番にコンパイルやリンクを実行してくれるツールもあったが，基本的にはそれぞれのツールの結びつきは薄かった．とはいえ，パソコンのレベルでも複雑で大きなプログラムが書ける環境が整ったという意義は大きかった．

統合環境付きCコンパイラの登場

Turbo CとQuickC

1987年には統合環境をもつCコンパイラが相次いで登場する．Turbo CとQuickCである．Turbo Cは，高速なコンパイル・リンクと，エディタを中心とするプルダウンメニューで機能選択のできる統合環境をもち，そのうえ99.95ドルという低価格なものであった．QuickCもTurbo Cと同じように，エディタ中心の統合環境をもつ低価格のCコンパイラだが，その同じ環境でソースレベルのデバッグを行うこともできた．

だが，この2つの処理系において実現された統合環境はまだ完全ではなかった．そこで扱えるプログラムは特定のメモリモデルに限られていたり，エディタで扱えるファイルの数が1つに

統合開発環境用語集
インデント　行の先頭にタブや空白を入れること．プログラミングスタイルのひとつでソースファイルでプログラムの流れを判断しやすくする．

限られていたりといった制限があったのだ．その不足した機能は，コマンドライン版のコンパイラを別プログラムとして付けるという形で補われていた．しかし，それらの欠点は，バージョンアップとともに着実に解消されつつある(表1)．

本格的な統合開発環境時代の到来へ

オブジェクト指向への流れ

Borland社はその後，PascalにもVer.4.0でCと同じエディタ中心，プルダウンメニュー方式のインターフェースを取り入れた．その後，同社の言語製品はオブジェクト指向へとシフトし，Turbo Pascal Ver.5.5でオブジェクト指向に対応，Turbo Cは新たにTurbo C++へと発展する．

オブジェクト指向プログラミングによれば，オブジェクトという考え方を導入することによって，プログラムはそれ自体「部品」として再利用可能になるという．すべての操作をひとつのプラットフォームで行える統合環境型のインターフェースは，この考え方とたいへん親和性が高いといえるだろう．

他方，もう一方の雄であるMicrosoft社はMS-C 6.0で，ついにProgrammer's WorkBench (PWB) という統合開発環境プラットフォームを登場させた．詳しい内容は特集のPWBの紹介記事に譲るが，MS-Cが統合開発環境を提供したことによって，今後，パーソナルなプログラム開発においては，エディタを中心とする統合環境型インターフェースが標準的な形となっていくのではないだろうか．

表1 バージョンアップの歴史

●Turbo Pascal

バージョン	特徴
1.0	統合環境をもったコンパイラ言語として登場
2.0	オーバーレイのサポート，全体的な機能向上
3.0	全体的な機能向上，グラフィックサポート（PC-9800シリーズ）
4.0	Turboシリーズのほかの言語とユーザーインターフェースを共通化，コマンドライン版コンパイラの追加，分割コンパイルのサポート（オーバーレイを廃止）
5.0	オーバーレイ機能を復活，全体的な機能向上
5.5	オブジェクト指向プログラミングをサポート
6.0	クラスライブラリTurbo Vision，オーバーラップウィンドウ

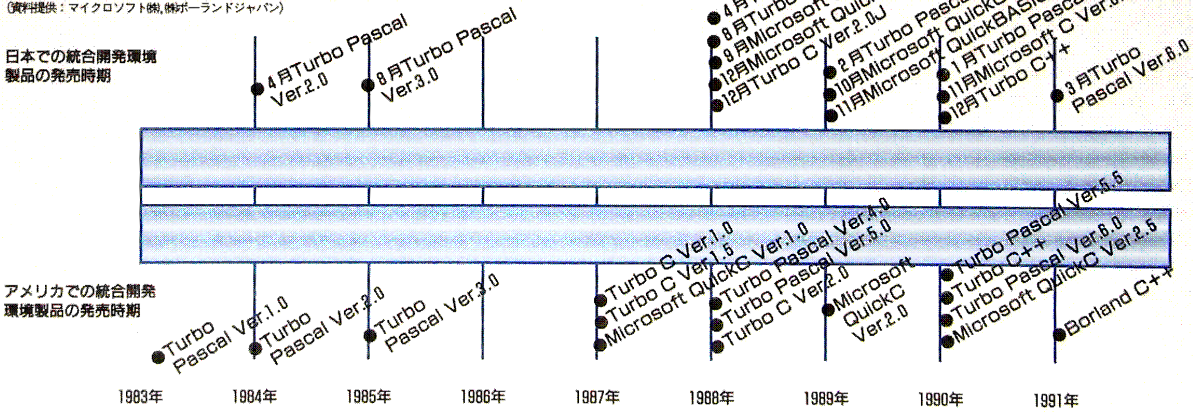
●Turbo C

バージョン	特徴
1.0	エディタ中心の統合環境を備えたCコンパイラとして登場
1.5	全体的な機能強化
2.0	ソースレベルデバッグ機能を統合環境に組み込み，エディタのEMS対応，グラフィックライブラリBGIのサポート，常駐型ヘルプユーティリティのサポートなど

●QuickC

バージョン	特徴
1.0	統合環境を備えた低価格Cコンパイラとして登場
2.0	MS-C5.1互換，統合環境でもミディアム以外のメモリモデルもサポート，インラインアセンブラが記述可能など
2.5	MS-C6.0互換（日本では未発売）

言語製品の流れとプログラミング環境の発展
(資料提供：マイクロソフト(株)，(株)ボーランドジャパン)



統合開発環境用語集
インラインアセンブラ　MS-CやTC++はソースファイル中に直接挿入したアセンブリコードを解釈しコンパイルする機能がある．

特集

研究「プログラミング統合環境」

快適で効率的な開発環境を考える

PWB/Quick/IDE

PC-9801用Cコンパイラの統合環境を比較する

統合環境をもつ言語処理系は数多く存在しているが，中でも最近注目を集めているのがC言語処理系である．これは，長い間統合環境というようなものをもたず，地味な処理系であったMS-Cが，6.0にバージョンアップされるとともにProgrammer's WorkBench（PWB）と呼ばれる強力な統合環境を備えて再登場したことによる．これを迎え撃つ統合環境の雄，Borland社もまたIntegrated Development Environment（IDE）と呼ばれるより強力な統合環境をもつTurbo C++を登場させた．ここでは，この2つの代表的なCコンパイラにマイクロソフト社がC言語の入門者用に発売しているQuickCを加えた3つの統合環境を，プログラマとのインターフェースを形成する画面を中心に比較，紹介していく．

PWB

(Programmer's WorkBench)

PWB は，MS-C 6.0（Microsoft C Professional Development System）が備えている統合環境である．PWB の操作は，マウスとウインドウを中心として構成されている．もちろんキーボードのみの使用も可能であるが，マウスを利用したほうが快適に操作できる．PWB の特徴は，Microsoft Editor を拡張したエディタを中心に，CodeView，NMAKE，リンカ，Microsoft Advisor などのツールを統合して利用できる点である．以下に PWB の画面と機能を示す．

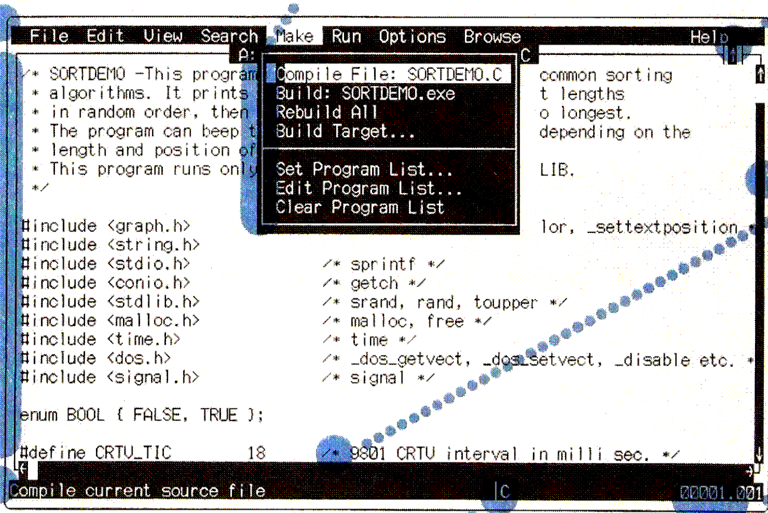
エディタ

PWB の中心である．ここから，CodeView や NMAKE，コンパイラを呼び出してプログラミングを行うことになる．

メニューバー
選択できるメニューを表示する．マウスでクリックするか，GRPH キーを押すことでメニューを選択できる．

プルダウンメニュー
メニューバーからメニューを選択するとプルダウンメニューが表示される．ここで表示しているのは MAKE に関するコマンド．

拡大ボタン
マウスでクリックするとそのウインドウが画面いっぱいに広がる．



ウインドウ
ソースファイルを表示して編集を行う．マルチファイルの編集が可能である．

スクロールバー
現在のカーソル位置を表示する．矢印の部分をマウスでクリックするとウインドウがスクロールする．

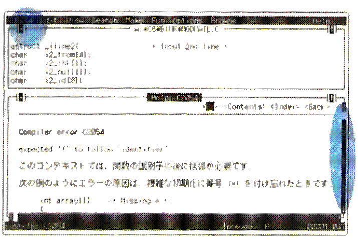
リファレンスバー
主要なコマンドやメッセージ，状態を表示する．

行/カラムインジケータ
現在のカーソル位置の行とカラムを表示する．

ファイルタイプインジケータ
カレントウインドウのファイルタイプを示す．

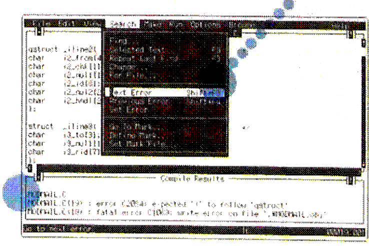
クローズボタン
マウスでクリックするとウインドウが閉じる．

プルダウンメニューから Next Error を選ぶか，Shift+f･3を押すと次のエラーにジャンプする．



f･1キーを押すと，状況に応じたヘルプが表示される．

コンパイル時にエラーがあると，自動的に表示される．



CodeView

高機能なソースレベルデバッガである．ブレークポイントやウォッチポイントの設定やトレース，ステップ実行など，さまざまな機能をもっている．

①メニューバー
選択できるメニューを表示する．マウスでクリックするか，GRPHキーを押すことでメニューを選択できる．

②ソースウインドウ
デバッグ対象のソースを表示する．

③クローズボタン
マウスでクリックするとウインドウが閉じる．

④拡大ボタン
マウスでクリックするとそのウインドウが画面いっぱいに広がる．

⑤スクロールバー
現在のカーソル位置を表示する．矢印の部分をマウスでクリックするとウインドウがスクロールする．

⑥リファレンスバー
主要なコマンドやメッセージ，状態を表示する．

⑦ローカルウインドウ
現在実行中の関数のすべてのローカル変数の内容を表示する．

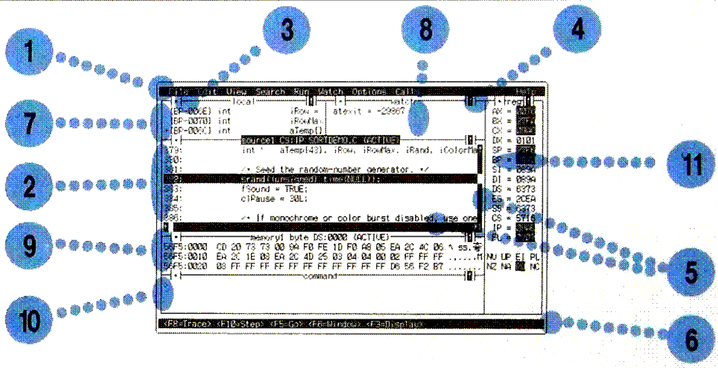
⑧ウォッチウインドウ
ウォッチ変数としてセットされている変数の値を表示する．構造体または配列の個々の要素を表示させることもできる．

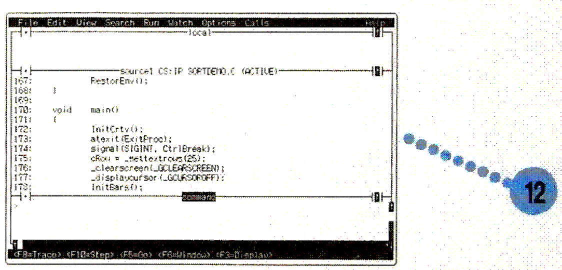
⑨メモリウインドウ
メモリの連続した領域を示す．

⑩コマンドウインドウ
コマンドの入力を行うウインドウ．

⑪レジスタウインドウ
現在のレジスタの状態を表示する．

⑫PWBからCodeViewを起動した直後の画面



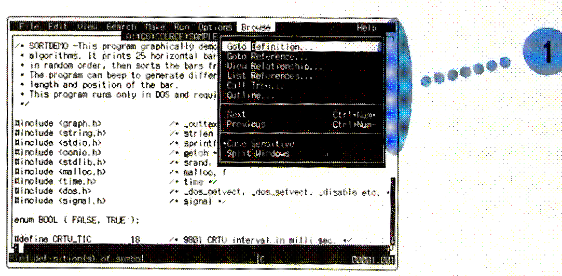


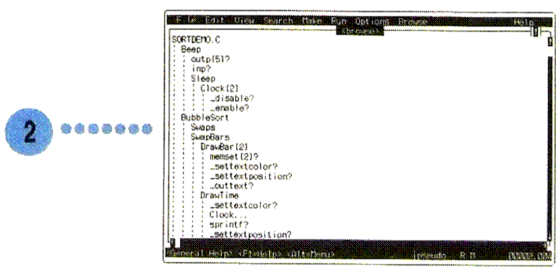
ブラウザ

関数と変数の関係やプログラムコードとディスプレイ情報をチェックすることのできるソースブラウザである．ブラウザは，エディタ上で利用できる．

①ブラウザ関連のコマンド
Goto Definition：インクルードファイルやモジュールの中にあるシンボルや変数が定義されているファイル名と行番号を表示する．
Goto Reference：インクルードファイルとモジュールでシンボルと変数が引用されたファイルと行番号のリストを表示する．
View Relationship：関数，マクロ，タイプ，変数などの関係を表示する．
The List References：関数，マクロ，タイプ，変数の引用を表示する．
Call Tree：プログラム内の関数コールのツリー構造を表示する．

②関数コールのツリー構造を表示





統合開発環境用語集
エスケープシーケンス　ソースプログラム作成時，文字列制御文字を記述するために使われる．0x1b(ESC)や0x5c('¥')に続く表記方法．

QuickC

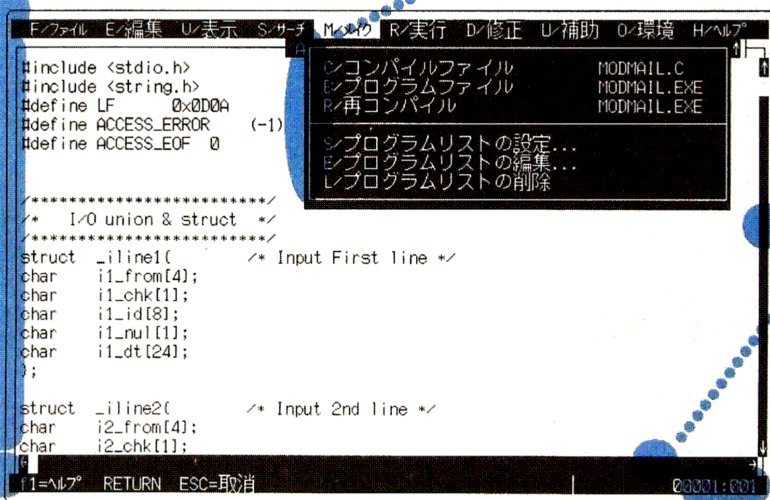
QuickC は，マイクロソフト社が C 言語入門者向けに発売している統合環境を備えた C コンパイラである．QuickC では，エディタ，コンパイラ，リンカ，デバッガが一体となっており，プログラマは QuickC 環境内でプログラミングに必要なすべての操作を行うことができる．

エディタ

WordStar ライクなコマンドを備えたシンプルなテキストエディタである．MKKEY ユーティリティを用いてカスタマイズすることも可能である．

メニューバー
使用可能なメニューを一覧表示する．

プルダウンメニュー
選択できるコマンドを表示する．ここでは，MAKE関連のプルダウンメニューが表示されている．

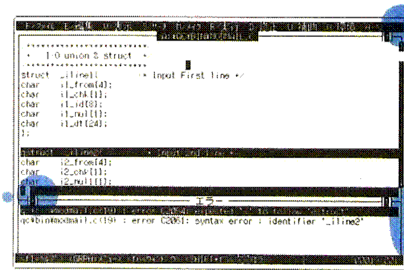


ソースウインドウ
現在編集しているソースコードを表示する．

スクロールバー
現在表示している部分が，ソースファイルのどの部分かを示す．矢印の部分をマウスでクリックするとその方向にスクロールする．

リファレンスバー
ショートカットキーやダイレクトコマンドを表示する．

行，桁インジケータ
カーソルの現在位置を行と桁で表示する．



サイズ変更
編集中のウインドウの大きさを変更する．

クローズボタン
編集中のウインドウを閉じる．ただし，ソースウインドウを閉じることはできない．

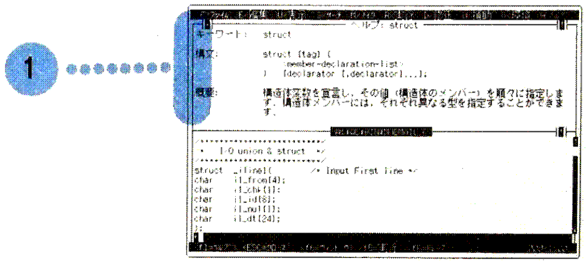
コンパイルエラーが生じると自動的にエラーメッセージが表示され，エラー位置にカーソルが移動する．

統合開発環境用語集
エミュレータ　数値演算コプロセッサと同じ処理をソフトウェアによって行う浮動小数点演算パッケージ．

QuickCアドバイザ

QuickC アドバイザは，C 言語の学習者がいちいちマニュアルをひく手間をはぶくために C 言語の文法やリファレンスを含んでいる．

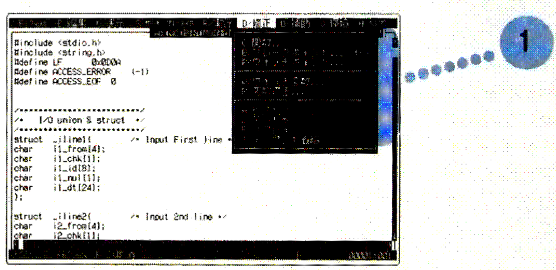
①キーワードヘルプ
　ヘルプを見たい単語の上にカーソルを合わせ f･1 キーを押すと，そのキーワードに関連したヘルプが表示される．

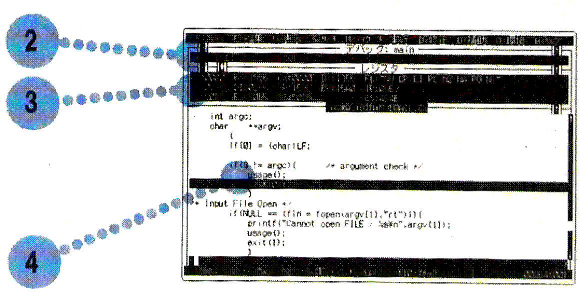


デバッガ

QuickC の内蔵デバッガでは，メニューのR／実行およびD／修正からコマンドを選択することで，トレースやステップ実行，ウォッチポイントやブレークポイントの設定などを行うことができる．

①メニューのD／修正に含まれるコマンド
②デバッグウインドウ
　指定した変数の内容を表示する．
③レジスタウインドウ
　現在のレジスタの内容を表示する．
④トレース実行中のカーソル





統合開発環境用語集
オーバーラップ　複数のウィンドウを重ね合わせて表示すること．

IDE

(Integrated Development Environment)

IDE は，従来のボーランド社 Turbo シリーズがもっていた統合開発環境をさらに発展させたもので，Programmer's Platform と呼ばれている．IDE では，ウインドウにオーバーラッピングウインドウを採用し，操作にはマウスを利用することができる．

また，Turbo Assembler や Turbo Debugger，Turbo Profiler などのツールを内部コマンドのように利用することが可能である．

エディタ

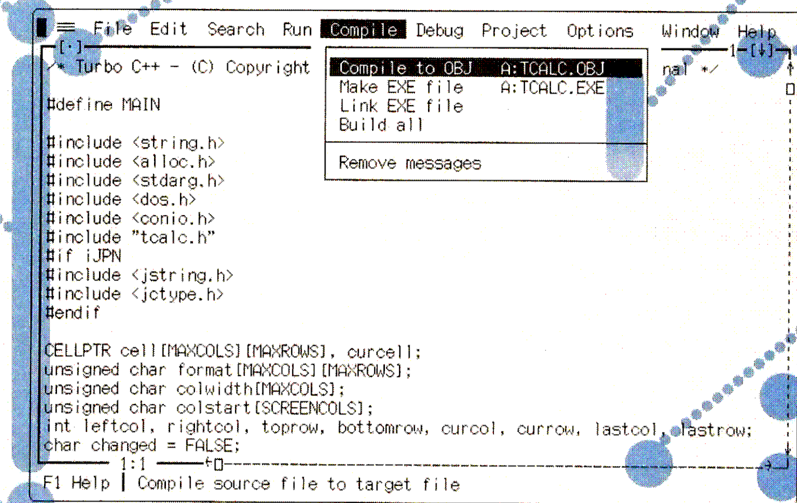
オーバーラッピングウインドウを採用したマルチファイル，マルチウインドウのエディタである．ファイルサイズは，ディスク容量のみに制限される．また，TEML（Turbo Editor Macro Language）と呼ばれるマクロ言語によるカスタマイズが可能である．

メニューバー
メニューバーはメニューコマンドに入る基点である．マウスでメニューバーのタイトルをクリックするか，f･10キーを押すことでメニューバーはアクティブになる．

クローズボックス
マウスでクリックするとウインドウが閉じる．

プルダウンメニュー
メニュータイトルを選択するとプルダウンメニューが表示される．ここで示しているのはコンパイル関連のコマンド．

ズームボックス
ウインドウの拡大・縮小を行う．



ウインドウ
ウインドウには，編集対象となるソースコードなどが表示される．ウインドウは，メモリが許す限りいくつでも開くことができる．ウインドウの形状は，オーバーラッピング，タイリングのどちらも可能．

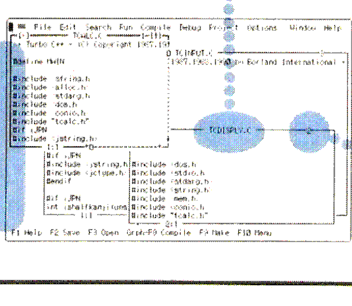
リサイズコーナー
マウスでドラッグしてウインドウのサイズを変更する．

スクロールバー
ウインドウに表示している内容をスクロールさせる．

クイックリファレンス行
基本的なキー操作やTurbo C++の状態を示す．

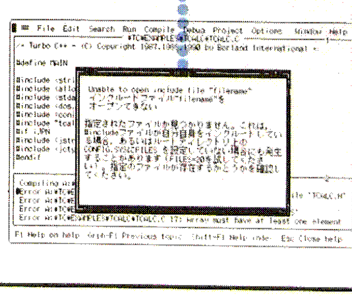
オンラインヘルプ
どのような状況でもf･1キーを押すことで，その状況にあったヘルプが示される．また，エディットウインドウで単語にカーソルを合わせ，CTRLキーとf･1キーを同時に押すとその単語に関するヘルプが示される．

タイトルバー
ウインドウの名前を表示する．



オーバーラッピングウインドウ
複数のウインドウを重ね合わせて表示することができる．

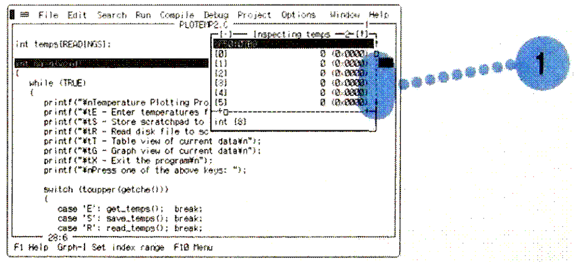
ウインドウ番号
GRPHキーとともにこの番号のキーを押すと，そのウインドウがアクティブになる．

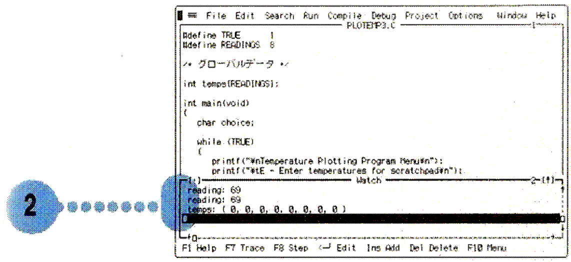


デバッガ

IDEには，トレース実行やブレークポイントのセットなどができるソースレベルデバッガが組み込まれている．

①Inspectorウインドウ
Inspector機能を利用すると，CまたはC++の有効な式の値を表示することができる．
②Watchウインドウ
ウォッチ式を設定して，変数の値を監視する．

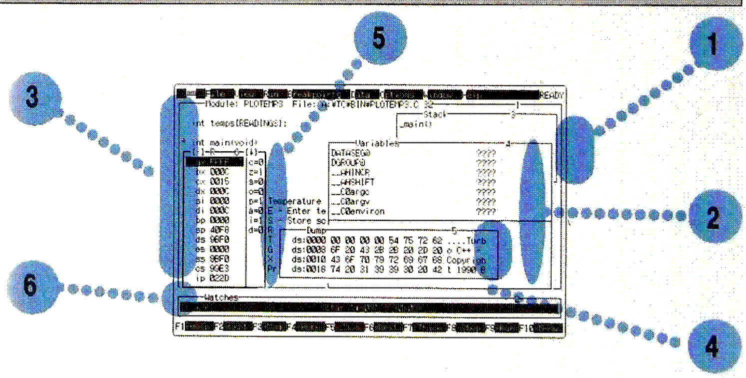


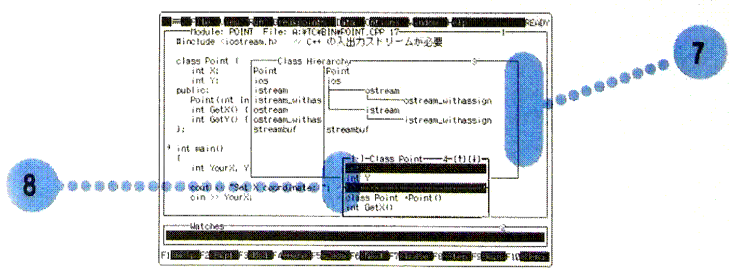


Turbo Debugger

Turbo Debuggerは，ボーランドのC，C++，Pascal，アセンブラで記述されたプログラムおよびCodeView形式のデバッグ情報を生成できるコンパイラで書かれたプログラムをデバッグできる高機能なデバッガである．

①Stackウインドウ
現在のスタックの状態を示す
②Variablesウインドウ
アクセス可能なすべての変数を表示する．
③Fileウインドウ
ディスクファイルの内容を表示する．
④Dumpウインドウ
メモリの指定された領域の内容を表示する．
⑤Registersウインドウ
CPUのレジスタとフラグの内容を表示する．
⑥Watchesウインドウ
変数とそれらの現在の値を表示する．
⑦Hierarchyウインドウ
現在のモジュールで使われているすべてのオブジェクトやクラスの型の階層ツリーを表示する．
⑧オブジェクト型／クラスInspectorウインドウ
オブジェクトの型の詳細を調べるためのもので，型情報の要約を表示する．





統合開発環境用語集
オブジェクト指向　オブジェクトに着目し，カプセル化・継承・多義性などによって各要素を抽象化するプログラミング手法

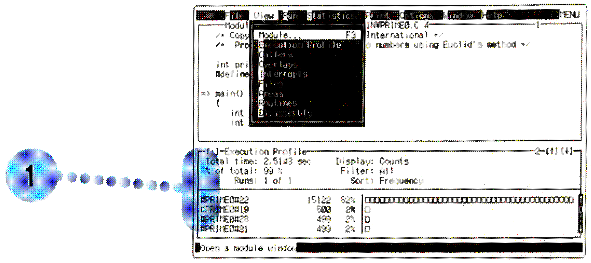
Turbo Profiler

Turbo Profilerは，プログラマが作成したプログラムの効率を詳細に解析し，そのボトルネックを捜し出すツールである．解析結果をもとにプログラムを改善することで，プログラムのパフォーマンスを向上させることができる．

①Execution Profileウインドウ

プログラムをプロファイルした測定結果が表示される．表示される内容は，プログラムの総実行時間，領域名とその領域で費やされる時間および総実行時間に体する割合など．

Turbo Profilerでは，関数，モジュールおよびプログラムの各行について，実行時間と実行回数を測定することも可能である．



●統合環境機能比較表

注）CV：CodeViewにより実現
TD：Turbo Debuggerにより実現

	Microsoft C 6.0	Borland Turbo C++ 1.0	Microsoft QuickC 2.0
□統合環境			
オーバーラップウィンドウ	×	○	×
マウスによる操作	○	○	○
8086/V30 CPUでの利用	×	○	○
入力ヒストリ	×	○	×
外部プログラムの呼び出し	○	○	○
メニューの日本語表示	×	○	○
ハイレゾ対応	×	○	×
OS/2対応	○	×	×
□デバッガ			
ソースレベルデバッグ	○(CV)	○	○
変数のウォッチ	○(CV)	○	○
ブレークポイント	○(CV)	○	○
条件式の指定	○(CV)	×	○
パスカウントの指定	○(CV)	×	○
レジスタウィンドウ	○(CV)	○	○
EMSを使った省メモリデバッグ	○(CV)	○(TD)	×
80386の仮想デバッグ	×	○(TD)	×
デバイスドライバのデバッグ	×	○(TD)	×
常駐プログラムのデバッグ	×	○(TD)	×
リモートデバッグ	×	○(TD)	×
ハイレゾ対応	×	○	×
OS/2対応	○	×	×
□エディタ			
オープン可能なファイル数	メモリに依存	メモリに依存	2
ウィンドウ分割数	8	メモリに依存	2
最大ファイルサイズ	メモリに依存	8MB	メモリに依存
検索・置換	○（複数で可）	○	○
正規表現	○	○	○
英数字の全半角同一視	×	○	×
範囲指定	×	○	×
方向指定	×	○	×
ブックマーク	○	×	○
アンドゥ／リドゥのレベル数	255	×	1
□コンパイラ			
メモリモデル	S/M/C/L/H/T	S/M/C/L/H/T	S/M/C/L/H
インラインアセンブラ	○	○	○
ANSI準拠度	75/79	77/79	63/79
アセンブリリストの出力	○	○	×
ハイレゾ対応	×	○	×
OS/2対応	○	×	×
□ライブラリ			
BIOSライブラリ	5	9	5
ハイレゾ対応	×	○	×
OS/2対応	○	×	×
□オーバーレイマネージャー			
EMSメモリの利用	×	○	×
プロテクトメモリの利用	×	○	×
関数へのポインタ	×	○	×
□ヘルプ機能			
ランゲージヘルプ	○	○	○
ライブラリ関数ヘルプ	○	○	○
サンプルプログラム	○	○	○
ハイレゾ対応	×	○	×
□そのほか			
オブジェクト指向プログラミング	×	○	×
ソースブラウザ	○	×	×

統合開発環境用語集

オプティマイズ　最適化．高級言語においてむだなコードを省き，サイズを小さく，または実行速度を速くするためにプログラムを加工すること．

特集 研究「プログラミング統合環境」

快適で効率的な開発環境を考える

PWBは標準ツールとなり得るか?

MS-C 6.0の統合開発環境

MS-Cの最新バージョン6.0では，従来からあるコマンドライン型のコンパイラ，リンカに加えて，QuickCライクな統合開発環境が取り入れられた．新しい統合開発環境の中心となるのはPWB（Programmer's WorkBench）である．PWBは強力なエディタを備え，かつコンパイラ，リンカ，MAKE，CodeViewデバッガなどのツールの機能を利用するためのプラットフォームでもあるが，QuickCよりもはるかにきめこまかい設定が可能で，ユーザー自身が機能を拡張することも可能である．このPWBを中心とする環境は，今後マイクロソフト社の言語製品での標準となっていくと予想される．ここでは，OS/2版を例に，PWBの諸機能と，PWBをカスタマイズする方法について紹介する．

メモリを気にせず使える OS/2上のPWB

海老原浩之

PWB（Programmer's WorkBench）は，Microsoft C Professional Development System Ver. 6.0（以下MS-C Ver.6.0と略す）に付属するユーティリティで，MS-C Ver. 6.0においてプログラム開発の統合環境を実現するものである．MS-Cは，ひとつのパッケージにOS/2版とMS-DOS版の2種類が含まれているが，ここではOS/2版を例に，PWBについて説明していくことにする．

動作環境

OS/2では基本メモリ＋1MBで

OS/2上でPWBを動作させるための環境については，表1にまとめたので参照していただきたい．この表でまとめた内容は，OS/2上でMS-C Ver. 6.0を構築する環境でもある．また，OS/2版のPWBはDOS版と違い，主メモリの空き容量やEMSの容量などを気にせずに使うことができる．OS/2で必要とする基本メモリ（4.5MB）に加えて，1MB以上のプロテクトメモリを積んでおけば，ふつうにPWBを使うことができる．

PWBの基本機能

OS/2でもDOSなみの操作環境

PWBの基本的な操作は，DOS版のQuickC Compiler Ver.2.0（以下QuickCと略す）とほぼ同じである．エディット，コンパイル，リンク，デバッグといった操作がひとつの統合環境として実現されているため，DOS版でQuickCを使い慣れている人には問題なく使えるであろう．

基本的な機能には，テキストエディタ，コンパイラ，リンカ，デバッガ，MAKEユーティリティ，ソースコードブラウザ，オンラインヘルプなどがある．これらの操作環境は，DOSにおいてはTurbo C，QuickCなどで，このごろではあたりまえに使われてきている環境である．しかし，今までこのような統合環境は，OS/2上ではなかったものであり，OS/2においてのプログラミング統合環境を実現する意味で今後は非常に期待できるものであるといえる．

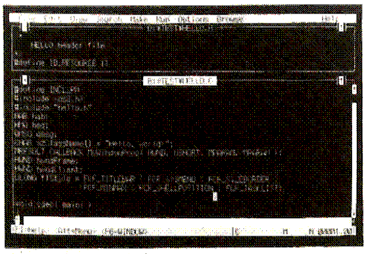
ここではまず，PWBの特徴であるエディタ機能，プログラムメンテナンス機能，ソースブラウザ機能，オンラインヘルプ機能など個々の機能について簡単に説明する．

◆エディタ機能

PWBの登場によって従来のOS/2 Ver.1.1のSDT（ソフトウェア開発ツール）やMS-C Ver.5.1に付属したエディタE.EXEがなくなった．それは，新しいエディタがE.EXEを機能拡張した形でPWBに組み込まれているためである．当然E.EXEで扱えたヘルプ機能もPWB上から使用することができる．また，マルチウィンドウ，マルチファイルで使えるため，最大で8個までウィンドウを開くことができ，それぞれのウィンドウごとにファイルをオープンすることができる．さらにキーの割り当て，ショートカットキーの割り当てなどでPWBのエディタキーアサインを変更することができる．加えてカスタマイズによって，エディタの操作を機能拡張することも可能である．すでに特定のエディタを使いなれている場合には，新たにPWBのエディタの操作方法を覚えるのは，なかなか面倒なものだ．それゆえPWBのエディタを使いなれたエディタに合わせてカスタマイズできるのは非常に便利な機能だといえるだろう．また，初期化ファイルTOOLS.INIの情報を変更すれば，エディタの基本機能も変更することが可能である．

・CPUが286以上のPC-9801シリーズ，PC-98シリーズ
・OS/2 Ver 1.1A以上
・メモリ4.5MB以上（空きメモリとして1MB以上が必要）（PWBとCodeViewを使うなら8MBは欲しい）
・12.5MB以上の空きがあるハードディスクとフロッピーディスクドライブ1基

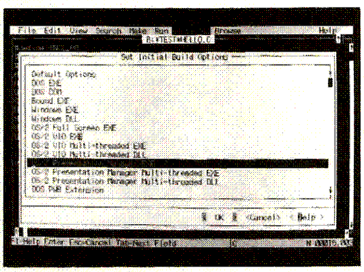
表1 OS/2上でPWBを使うための環境



画面1　複数ウィンドウ

◆プログラムメンテナンス機能

QuickC Ver.2.0と同様のプログラムメンテナンス機能を備えている．MAKEファイルの自動生成やターゲットのソースに依存するファイルの自動検索，BuildオプションによるOS/2のDLLやPMプログラムなどのコンパイル，リンクのオプション参照，登録などがある．これらの機能によりOS/2上における複雑なプログラムの開発も楽になるはずである．

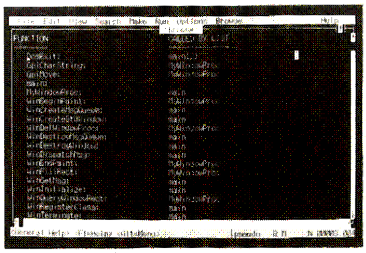


画面2　Buildオプション

◆ソースブラウザ機能

作成中のソフトウェアの構造を解析する機能である．プログラムをコンパイル中にデータベースを作成するもので，作成されたデータベースをもとに対話形式で情報を検索することが可能である．ソースブラウザの機能として次のものがあげられる．

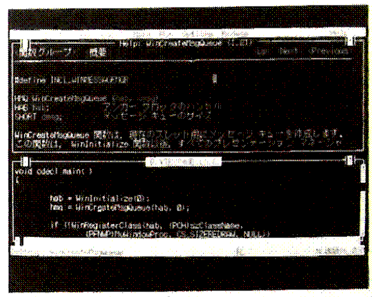
- ●関数，変数，定数の定義部分への移動
- ●関数，変数，定数の関係表示
- ●関数や変数への参照と検索一覧表示
- ●プログラムの構造を表示
- ●指定した関数，または関数を使用する関数の表示



画面3　ソースブラウザ機能

◆オンラインヘルプ機能

MS-C Ver.6.0の言語仕様，ライブラリの説明，PWBの操作方法説明などマニュアルに匹敵する内容をオンラインで参照できる．これにより，従来のMS-C Ver.5.1などのようにぶ厚いマニュアルを見なくてもすむようになっている．



画面4　オンラインヘルプ

環境設定

環境変数と初期化ファイル

PWBを使うためには環境設定が重要である．まず環境変数だが，2つの環境変数をセットしなくてはならない．1つは，INIT環境変数，もう1つはHELPFILES環境変数である．この2

リスト1　CONFIG.SYS

```
protshell=B:¥OS2¥PMSHELL.EXE B:¥OS2¥OS2.INI B:¥OS2¥OS2SYS.INI B:¥OS2¥CMD.EXE /K B:¥OS2¥OS2INIT.CMD
set comspec=B:¥OS2¥CMD.EXE
libpath=B:¥OS2¥DLL;B:¥;b:¥sdt¥help;b:¥sdt¥samples¥dll;
set path=a:¥C6¥BINB;a:¥C6¥BINP;B:¥OS2;B:¥OS2¥SYSTEM;B:¥;B:¥OS2¥INTRO;b:¥sdt¥binb;b:¥sdt¥binp;b:¥sdt¥
binp¥ipf;b:¥sdt¥help;
set dpath=b:¥sdt¥binp¥ipf;B:¥OS2;B:¥OS2¥SYSTEM;B:¥;
set prompt=$i[$p]
set include=a:¥C6¥INCLUDE;b:¥sdt¥include
set lib=a:¥C6¥LIB;b:¥sdt¥lib;b:¥sdt¥samples¥lib;
set helpfiles=a:¥C6¥HELP¥*.HLP;b:¥sdt¥help¥*.hlp;b:¥sdt¥binp¥ipf¥*.hlp;b:¥sdt¥binp¥*.hlp
set ipfc=b:¥sdt¥binp¥ipf
set help=B:¥OS2¥HELP;b:¥sdt¥binp;
set init=a:¥C6¥INIT;b:¥sdt¥binp¥init
set bookshelf=B:¥OS2¥BOOK
SET TMP=b:¥
buffers=50
diskcache=64
maxwait=3
memman=SWAP,MOVE,NOSWAPDOS
swappath=B:¥OS2¥SYSTEM 512
protectonly=NO
iopl=NO
threads=255
break=OFF
fcbs=16,8
shell=B:¥OS2¥COMMAND.COM ¥OS2 /P
device=B:¥OS2¥DOS.SYS
country=081,B:¥OS2¥SYSTEM¥COUNTRY.SYS
device=B:¥OS2¥VDISK.SYS 256 512 128
device=B:¥OS2¥DRV3X01.SYS
device=c:¥sys@dev¥atok7as.sys /D=C:¥ATOK7L.DIC
device=c:¥sys@dev¥atok7bs.sys
device=B:¥OS2¥DRV3X02.SYS
device=B:¥OS2¥MOUSE01.SYS MODE=P
device=B:¥OS2¥POINTDD.SYS
rem device=B:¥OS2¥COM01.SYS
run=B:¥OS2¥SYSTEM¥PRNCONV.EXE
codepage=932,437
device=B:¥OS2¥PMDD.SYS /CS
devinfo=KBD,JP,B:¥OS2¥KEYBOARD.DCP
run=B:¥OS2¥IMSTART.EXE B:¥OS2¥IMMON.EXE @B:¥OS2¥IMINFO.PRO
set keys=ON
```


統合開発環境用語集
カーソル　マウスなどのポインティングデバイスの指す位置を示すシンボル．

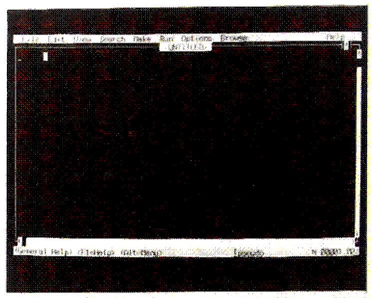
つの環境変数なくしてはPWBは正常に動作しない．

INITはTOOLS.INIファイルが格納されているディレクトリを指し，HELPFILESはPWBやQuick-HELPなどのヘルプファイルが格納されているディレクトリを指す．INIT環境変数がセットされていない場合は，本来ならINITディレクトリにあるファイルがカレントディレクトリにあるものとみなされ，HELP-FILES環境変数がセットされてないとPWB上からヘルプが参照できなくなってしまうので，この2つの環境変数はしっかりとCONFIG.SYSにセットするようにする（リスト1）．

次にTOOLS.INIとCURRENT.STSの2つのファイルについて簡単に説明する．

TOOLS.INIファイルは，MS-C Ver.6.0がインストールされた直後は存在せず，C6¥INITのサブディレクトリにTOOLS.PREというファイル名で存在している．このTOOLS.PREファイルをTOOLS.INIにリネームするか，コピーしてTOOLS.INIとして使用する．TOOLS.INIファイルには，すべてのツール類（NMAKE, PWB, CodeViewなど）の情報が格納されている．PWBは，このファイルの情報にしたがって設定を行い，一度設定されたTOOLS.INIは，以後この内容がデフォルト値として利用される．また，TOOLS.INIファイルはテキストファイルなので，ユーザーがエディタなどで簡単に修正することもできるが，変な形に修正した場合はPWBが起動できなくなる恐れがあるのでよく注意する必要がある．

PWBを起動した後に作成されるファイルとしてCURRENT.STSファイルがある．このファイルは，最後にPWBを利用したときの情報が格納されており，次回の起動時に読み込まれて，ユーザーが前回PWBを使用した最後の状態でPWB画面が再現される．CURRENT.STSファイルもテキストファイルなので，ユーザーがエディタなどで修正することができる．


画面5　PWBの起動画面

PWBの操作
コンパイル，リンクと実行方法

PWBの起動方法だが，コマンドプロンプトから

```
[C:\]  PWB
```

で起動する．

この時点からすぐにでもプログラムの入力を開始できる．また，既存のプログラムファイルを読み込む場合は，「File」メニューをマウスでクリックし，プルダウンメニューからさらに「Open...」をマウスでクリックするとファイル名の入力ボックスが表示されるので，目的のプログラムファイルを選択して読み込む．また，マウスによる操作だけでなく，キーボードによる操作も可能である．たとえば，［GRPH］＋［F］キーで「File」メニューになる．このようなキーボード操作は，MS-Windowsとよく似ている．

次にPWB起動時のオプションだが，これは表2のようになっている．

MS-C Ver.6.0についてくるマニ

表2　PWBの起動オプション

PWBの起動

PWB /D [S | T | A]　[/e]　[/m mark_file]　[/PF make_file]　[/PL]　[/PP programlist_file]　[/r]　[/t file_name]　[/?]

オプションとその意味

/DS	CURRENT.STSの内容を無視する
/DT	TOOLS.INIの内容を無視する
/DA	PWB拡張機能をロードしない
/D	TOOLS.INI，CURRENT.STSの両方を無視しPWB拡張機能をロードしない
/e	起動時に実行するPWBの関数を指定する
/m	マークファイルで指定された位置にカーソルを移動して起動する
/PF	指定したメイクファイルを起動時に読み込む
/PL	以前にPWBが最後にオープンしていたプログラムリストを読み込む
/PP	起動時に指定したファイルのプログラムリストを読み込む
/r	読みだし専用モードで起動する
/t	起動時にロードするファイルを指定する
/?	PWBの起動時のオプション一覧を表示する

統合開発環境用語集
キャレット　マウスカーソルと区別するために文字カーソルをキャレットという．

ュアルには，OS/2上におけるPWBを使ったアプリケーション開発の方法がほとんどのっていないに等しい．これでは，実際にPMアプリケーションの開発にあたって，PWB上ではどのようにしたらいいかわからないと思っている人も多いことだろう．そこで，ここでは実際にNECのOS/2 Ver.1.21のSDTに付属するサンプルソースを使ってPWBの操作を追ってみることにした．ここで取りあげるサンプルソースとして，PM上で“HELLO”の文字を表示するHELLO.Cを例にする．

◆PWBの起動とソースプログラムの読み込み

まず，PWBを起動する．起動の直後には，画面にHELLO.Cは表示されない．まず「File」メニューから「Open...」を選択する（画面6）．File Listに「HELLO.C」が表示されるので，マウスカーソルを「HELLO.C」にもっていきダブルクリックする（画面7）．これで画面にHELLO.Cのプログラムソースが表示される（画面8）．

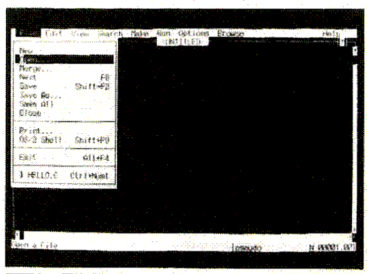
◆オプションの選択と確認

メニューバーから「Options」を選ぶ（画面9）．ここでは各種のオプションの設定と確認を行う．まず，「Environment...」で環境の確認を行う．ここは，コンパイルする場合のInclude File /Library FileのSerch PathとHelp FileのPathの確認ならびに設定を行うところである(画面10)．基本的にはCONFIG.SYSでSET INCLUDE, SET LIB, SET HELPの環境変数の設定さえしっかりしてあれば問題はない．もしも設定してない場合は，ここで設定すればよい．

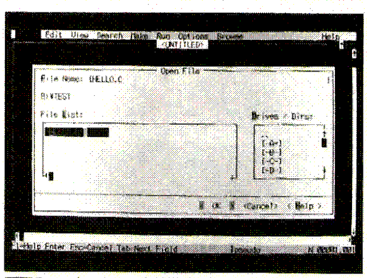
次に，「Build Options...」を選択する．ここではOS/2 PMのプログラム作成の基本的な設定が簡単に行える．まず，「Set Main Language」だが，この設定はC言語で作成した場合は，「C」とすればよい．その次に「Set Initial Build Options」を選ぶ．この中にはコンパイルやリンクに必要な設定情報（コンパイルスイッチやリンクスイッチなど）が，いろいろな開発状況に応じて用意されている（画面11）．今回はOS/2 PM用のプログラムの作成になるので，「OS/2 Presentation Manager EXE」を選択する．これでPM用プログラム作成の諸設定がPWBにセットされる．以下，Compiler Option/LINK Option/NMAKE Option/CodeView Option類に関しては，その時にコンパイルするプログラム状況によって多少異なってくるが，基本的には「Set Initial Build Options」で設定した内容でコンパイルとリンクを問題なく行ってくれる．特に設定する必要がある場合以外は，変更しなくてよい．また，変更した場合は「Save Current Build Options...」をクリックして情報を保存することができる．次回からは，保存した情報を使えばよい．Build Optionによって開発環境の設定はかなり便利になったといえるだろう．

◆メイクファイルの作成と実行

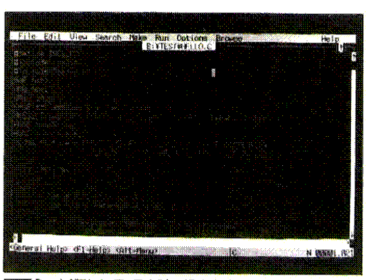
さてOptionで環境の設定が終わったら，メイクファイルの作成と実行である．メニューバーから「Make」を選択する(画面12)．ここで，HELLO.Cプログラムのメイクファイルを作成する．はじめに「Set Program List...」でコンパイルとリンクを行うプログラ



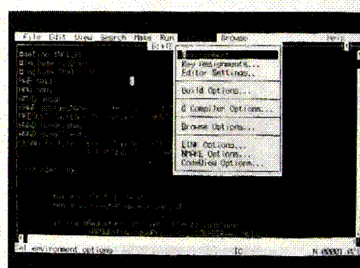
画面6　FILE メニュー OPEN



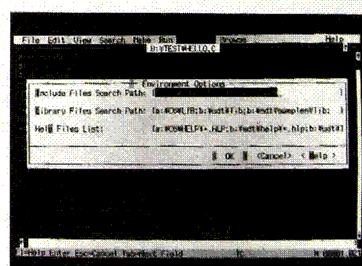
画面7　マウスカーソルがHELLO.Cに



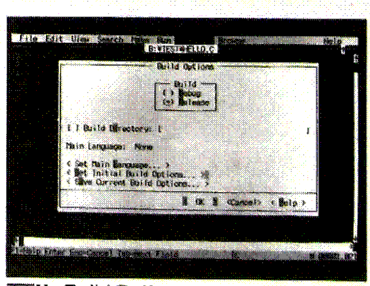
画面8　HELLO.Cを読み込んだところ



画面9　Optionsメニュー



画面10　Optionsメニュー-Environment



画面11　Build Options...

統合開発環境用語集
クリップボード　文字列の複写や移動において，対象となる文字列ブロックなどを格納するためのバッファのこと．

ムを指定する．画面13をみていただこう．何も表示されておらずFile Nameのところは「＊．mak」となってるのみである．ここで「HELLO.MAK」と入力すると，画面14のように「HELLO.MAK」というメイクファイルを作成するかどうか聞いてくるので「YES」を選択する．次に「Edit Program List」の画面（画面15）になる．ここで，コンパイルするプログラムを指定するわけだが，File Nameのところは「＊．C」となってるため，「File　List」のウィンドウには，「HELLO.C」しか表示されていない．このままではコンパイルとリンクを行ってもPMで実行できるプログラムは作成されない．そこで，必要なファイルを選択するためにFile Nameのところで，「＊．＊」と入力する．ここで，コンパイルに必要なファイルを「File List」のウィンドウの中から選択し，「Program List」に追加していく．追加の仕方は，必要なプログラム名をマウスでクリックし，「Add/Delete」をマウスでクリックするだけでよい．この場合は，HELLO.CとHELLO.RCの２つを選択する（画面16）．

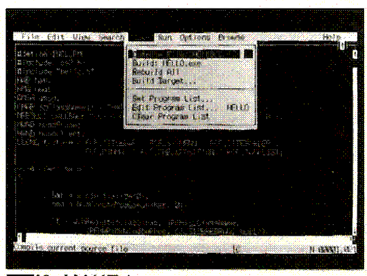
さて，ファイルの選択が終了したら，「Save List」により情報を保存する．これでHELLO.Cのメイクファイルが作成されたわけである．あとは「Rebuild All」を選択してmakeを実行する．

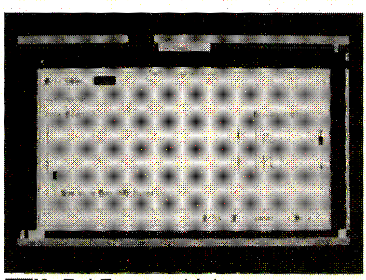
最後にHELLO.Cのコンパイルとリンクが完了すると，警告が４つほど（画面17）でるが，とくに問題はないのでできあがったHELLO.EXEを実行してみて欲しい．問題なくHELLO.EXEが動くはずである．

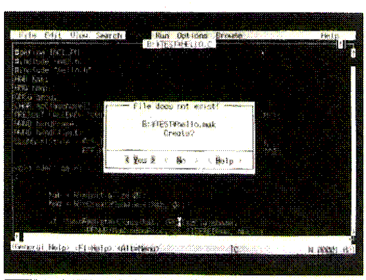
以上，簡単にPWBの操作を追ってみた．今回は紙面の関係もありコンパイルとリンク，プログラムの実行までの操作に絞らせてもらったが，これで多少なりともPWBの操作がわかってもらえたと思う．

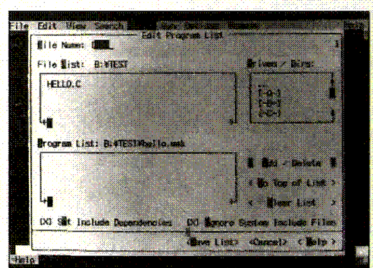
これまでPWBに関することを簡単に説明してきたわけだが，実際の操作には思ったより複雑さを感じる．PWBをはじめて使う人にとっては，使いなれるまで時間がかかるだろう．その理由のひとつはマニュアルである．操作方法などがオンラインヘルプで参照できるようになっているのはよいが，オンラインマニュアルに片寄りすぎて，実際のマニュアルでも詳しく操作については説明していないし，PWBの操作例がDOSのみのため，OS/2で扱う人にとってはちょっと難がある．とくにPMアプリケーションやDLLの開発などについては，ほとんど記述がない．

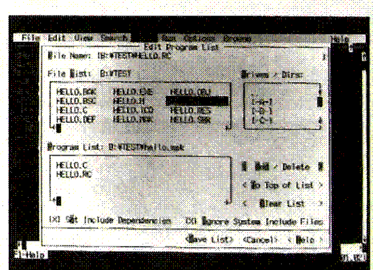
また，機能を豊富にしたためかどうかはわからないが，PWB自体動作が機敏でない．実際にPWB上でプログラム開発を行うには非常にスピード感が欠けていると思う．だが，OS/2上ではじめて実現した統合環境は評価に値すると思うし，スピードさえあがれば決して悪い環境ではない．また，私の個人的な意見として，PWBに関しては詳しい操作方法を述べた別マニュアルを作成してもらったほうが逆に親切のように思える．PWBについてはまだまだ改良の余地は残されていると思えるので，今後さらに改良されより使いやすい統合環境ツールとして発展してもらいたいものである．

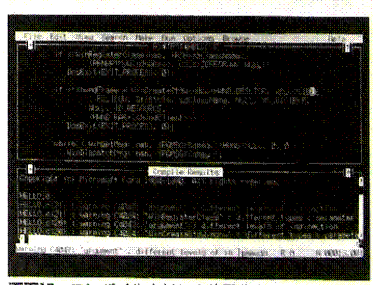

画面12　MAKEメニュー


画面13　Set Program List


画面14　HELLO.MAKを作成するかどうかきいてくる


画面15　Edit Program List


画面16　HELLO.CとHELLO.RCの２つを選択する


画面17　コンパイルとリンク後警告がでる

統合開発環境用語集
高級言語　CPUの構造に束縛されない言語．アセンブリ言語のようにCPUよりでなく，自然言語に近い．

PWBのカスタマイズ

つけ加えたい機能をCでプログラムする

種村明宏

PWBは,「スイッチ」「キー割り当て」「マクロの記述」「PWB自体の機能拡張」の4点について，ユーザーが独自にカスタマイズできるよう設計されています．最初の3点については，統合プログラミング環境を備えた製品としては，もはやあたりまえの機能だといえます．これに対して，PWB自体の機能拡張とは，ユーザーがPWBにつけ加えたい機能をC言語でプログラミングし，これをコンパイル，リンクしてPWBから呼び出せるようにするというものです．

ただし，これらの拡張機能についての詳細は，PWBのマニュアルの大部分がオンライン化されていることもあってわかりにくくなっています．ここでは，PWBの豊富なカスタマイズ機能を説明しながら，PWBの拡張方法を探っていきたいと思います．

PWBはOS/2で

6～8MBのメモリでようやく快適に

あらかじめお断りしておきたいのは，PWBをカスタマイズし，機能を拡張するにはそれなりのハードウェアが必要だということです．とくに，MS-DOS上で使うことを考えると，640KBという限られたメモリ空間では，対象となるプログラムのメモリ空間を圧迫してしまいますので，あまり大きな機能拡張をするのは考えものです．実際のところ，MS-C 6.0を購入して，DOS上でPWBを起動された方の多くは，「起動までの時間が長い」「重い」といった印象をもったのではないでしょうか．まったく拡張しない場合でも，そんな具合ですから，大きさや機能拡張時の制限を考えますと，PWBの拡張機能はDOSで使うよりは，むしろOS/2で使うものだとわりきって考えたほうがいいでしょう．

OS/2環境であれば，仮想メモリを使用できますので，PWB自体が使うメモリの量が増えても，メモリがたりなくなるという事態はおこりません．極端な例ですが，実メモリ4.6MBの環境で3つのPWB（うち1つでコンパイル），MIFES/2，PMアプリケーション，ウインドウコマンドプロンプト，フルスクリーンプロンプトを起動することが可能です．もちろん，この場合はメモリスワッピングが起こりますので（OS/2 V1.10で3.8MBのメモリスワッピング）その分，処理は遅くなります．

同じメモリ容量のOS/2 V1.21上で実行したところ，PWB+フルスクリーンシェルでは約1.7MBのメモリスワッピングが起きました．コンパイルを行ったり，他のツールをたちあげた場合には，2.7～3.2MBほどのメモリスワッピングになります．PWBの機能拡張や，CodeViewでのデバッグ，ほかのツールを「快適」に使うことを考えると，やはり6～8MBのメモリがあったほうがよいと思います．

スイッチの変更

各種の設定に有効

PWBでは，ユーザーが環境を独自に変更できるよう，多数のスイッチが用意されています．スイッチのカスタマイズの方法は2通りあります．1つめの方法は，PWBの「Options」メニューの「Editor Settings...」を選択するものです．疑似ファイルASSIGNがオープンされて，写真1のような画面が表示されます．スイッチには数字を代入する（Numeric Switch）と，yes/noで答える（Boolean Switch），語句でスイッチを設定する（Text Switch）の3種類があります．

以下にそれぞれのスイッチの例をあげます．

［例1］Numericスイッチ

```
height:25
```

heightスイッチは，ウィンドウの高さを制御します．通常ノーマルモード（640×400ドット）では最大値は25です．そのうち，ウィンドウで使用できる実際の表示領域の高さは21行となります．マイクロソフト版のOS/2上では，ハイレゾモードでPWBを使用することができます．このときheightは，最大値を31に設定することが可能です．

統合開発環境用語集
構造化　プログラムの制御が上位のモジュールから下位のモジュールに渡される階層構造をとるプログラミング手法

［例２］Booleanスイッチ

noの場合	yesの場合
autosave:no	autosave:yes
または	または
noautosave:	autosave:

autosaveをオンにすると，カレントファイルの編集を中断して他のファイルの編集を行うとき，および編集を終了させるときに自動的にファイルが保存されます．デフォルトではnoになっています．

［例３］Textスイッチ

```
build:macro  LINKER  "link"
```

buildスイッチは，ファイルをコンパイルするときにPWBが使う情報を定義します．次のような場合には，buildスイッチを設定する必要があります．

・「Options」メニューにないコンパイラ（言語）を使うとき

・新しいタイプのファイルをコンパイルするとき

例３では，macroを使ってリンカを「link.exe」に定義しています．NMAKEでは，「LINKER = link」となります．build:macroの書式を使うことによって，プログラムをメイクするユーティリティNMAKEに対して，さまざまなマクロを設定することができます．このスイッチの変更で編集する疑似ファイル「ASSIGN」は，実際にディスク中に存在するファイルではなく，PWBのメモリ上におかれて動的に変化するファイルです．

そのため，ファイルの設定を変更するとただちにその変更が有効になります．デフォルト値を変更して，ほかの行へカーソルを移動すると，変更した項目が黄色で強調表示されます．設定方法が違っていたり，まちがった値を代入した場合，その行は強調表示されず，エラーメッセージが出ます．

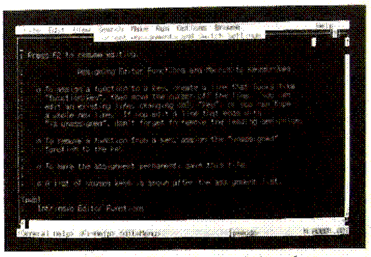
ただし，これだけでは設定を記憶しておくことはできません．設定を次回以降も有効にしたいときは，「FILE」メニューから「SAVE」を選択して，TOOLS.INIというファイルに書き込む必要があります．TOOLS.INIは，カスタマイズした内容を保持するPWBの初期化ファイルです．

スイッチを変更するもう１つの方法としては，このTOOLS.INIを直接書き換える方法です．はじめは，TOOLS.INIにはほとんど何も書かれていないので，スイッチの変更を行うだけなら，疑似ファイルを変更してSAVEするほうが便利です．疑似ファイルにない機能を追加するような場合（マクロやPWBの拡張機能を使用したユーザー関数の登録など，詳しくは後述）に，直接TOOLS.INIを書き換えます．TOOLS.INIファイルのユーザーの変更部分が削除された場合，PWBはデフォルトの値を使用します．

また，疑似ファイルの内容はプリントアウトもできますので，必要に応じて印刷して確認するとよいでしょう．

キーの割り当て
デフォルト値は残る

PWBでは，利用するキーの割り当てを設定することができます．ただし，設定前のデフォルトのキー設定もそのまま残ります．つまり，ユーザーがカスタマイズしても，PWBのもともとのキー割り当ては変わらないのです．


写真１　疑似ファイルASSIGNのオープン

キー割り当ての変更は，スイッチの変更の場合と同じく，「Options」メニューの「Key Assignments...」を選択して疑似ファイルを変更するか，またはTOOLS.INIファイルを直接編集します．

使用する疑似ファイルは，スイッチ変更のときと同じ「ASSIGN」で，変更の方法もスイッチの場合と同じです．キー名は，大文字小文字を区別しません．たとえば［Alt］+［C］と［Alt］+［c］は同じ意味になります．疑似ファイルを変更し，保存するとユーザーが変更したキーの割り当て部分のみがTOOLS.INIファイルに書き込まれます．TOOLS.INIのキー定義を削除すれば，カスタマイズは無効になります．

ただし，疑似ファイルの変更後，再度疑似ファイルを見るとデフォルトのキー割り当ての下に，ユーザーが定義したキー割り当てが表示されます．たしかに，操作性の統一という点では評価できますが，デフォルトでPWBが使用しているキーをほかの機能に割り当てることができないのは少々不便です．

スイッチの変更やキーの割り当てといった，TOOLS.INIファイルを使ったカスタマイズ方法は，共通のハードディスクに入っているMS-Cを複数のプログラマが共用する場合に有効でしょう．自分用のTOOLS.INIを使っ

統合開発環境用語集
最適化　高級言語においてむだなコードを省き，サイズを小さく，または実行速度を速くするためにプログラムを加工すること．

てPWBを起動すれば，それぞれが好みの環境を利用できることになります．

マクロによる機能追加

定義して保存する

マクロによって，PWBに編集機能を追加することもできます．マクロはPWB自体の機能，文字列，マクロ用演算子を組み合わせて作ります．マクロを定義するには次の方法があります．

・一時的にマクロを定義する
・一連のキー操作を記録してマクロとする

◆一時マクロの定義

[Alt]＋[A] を押すとリファレンスバー左に“Arg[1]”の文字が表示されます．ここで文字を入力すると，ダイアログボックスが開きます（写真2）．以下，この操作を[Alt]＋[A]操作と呼びます．

マクロの定義の書式は次のとおりです．

書式：マクロネーム：＝（関数，文字列，マクロなど）

入力が完了したら，[Alt]＋ [-] でダイアログボックスから抜けてください．このとき，テンキーの [-] キーは使用できないことに注意してください．

関数の引数

関数に引数が必要な場合は，例4のように記述します．Arg関数は，引数30をQreplace関数に渡します．このマクロを実行すると，カレント行から数えて，30行以内の間で検索置換を行います．実行されるとダイアログボックスが開いて，検索する文字列，それを置換する文字列を入力するように指示してきます．

［例4］関数の引数

```
qrepn:=Arg "30" Qreplace
```

マクロに対する応答

PWBで（yes/no）の応答が必要な機能の関数をマクロで実行すると，この応答ができません．必要な場合は関数の後に演算子をつける必要があります（表1）．

条件マクロ

条件に応じていろいろな働きをするマクロを記述することができます（表2）．PWBの関数は一般に終了時に成功ならTRUE(!=0)，失敗ならFALSE

表1 マクロに対する応答を表す演算子

演算子	意味
<	確認を求める この後のマクロに<演算子がなければ，これ以降の関数はすべて応答を求める
<y	yesの応答があったことになる
<n	noの応答があったことになる

表2 条件マクロ

演算子	意味
:>label	labelを定義する
=>label	labelにジャンプする
+>label	前の機能が返した値がTRUEなら，labelにジャンプする
->label	前の機能が返した値がFALSEなら，labelにジャンプする

統合開発環境用語集
シフトJIS　JIS漢字コードをシフトし，グラフィック文字部に割り当てたコード体系．MS-DOSや一部のUNIXで採用されている．

(=0) を返します.

◆一時マクロの実行

定義されたマクロは次の2つの方法で実行できます.

Execute関数の実行

[Alt]+[A]操作に続いて，定義したマクロネームを入力し，[F7]（Execute関数が割り当てられています）を押すことにより実行できます.

キーにマクロ機能を割り当てる

[Alt]+[A]操作に続いてマクロを入力します.

書式：マクロネーム：割り当てるキーの名前

[Alt]+[-]で抜けると定義終了です．定義されたキー割り当ては，疑似ファイル「ASSIGN」で参照することができます.

エディター内で常時使うマクロはキーに定義し，たまに使うマクロや危険を伴うマクロはExecute関数で実行したほうがいいでしょう．マクロで使用する関数には，PWBで提供される関数のほかに，PWBの拡張機能を利用したユーザー定義の関数も使うことができます（後述).

◆一時マクロの保存

一時マクロで作成したマクロは，PWBを終了させると消えてしまいます．定義したマクロを保存するには，TOOLS.INIの[PWB]セクションにマクロを定義しなければなりません.

◆一連のキー操作の記録

キー操作を記録するには，まず「Edit」メニューから「Record On」を選択します．ダイアログボックスが現れて，記録するマクロ名を入力するように指示してきます．デフォルト名はrecordvalueですが，ユーザーの好きな名前をつけられます．ここでは，testという名前をつけておきます．いったんつけた名前を変更したいときには，メニューから「Set Record...」を選択してください.

次に，マクロtestを呼び出すキーを割り当てます．割り当てるキーは，PWBで予約されているものでなければ何でもかまいません．ここでは[Alt]+[I]を割り当てることにしましょう．もしもこのとき，すでに同じキーがほかのマクロに割り当てられてるた場合には，それを無視するかどうかも設定することができます．すべての項目を設定し終えたら「OK」を選択してください．これで準備完了です.

実際にマクロに登録したいキー操作を行います．ここでは例として，次のように入力してみます.

```
[CTRL]+ [N]
/* Test Dayoon */
```

次に，メニューから再度「Record On」を選択します．このときには記録中であることを示す「*」マークが表示されているはずです．これで記録が完了しました．[Alt]+[I]キーを押せば，記録されたマクロを呼び出すことができます.

ところで，記録されたマクロはどこへいったのでしょうか？　メニューから「Edit Macro」を選択してください．さきほど記録したマクロが写真3のようにに入っているのがわかるでしょう．このファイルは，前述の「ASSIGN」と同じ疑似ファイルですから，このままでは，さきほどのマクロはPWBを終了した時点で消えてしまいます．記録したマクロを保存するには，TOOLS.INIファイルを開いて[PWB]のセクションにコピー&ペーストします．このときに，マクロtestの定義の後に，呼び出すためのキーの定義も書き込んでおきます.

この例では，次の1行を追加することになります.

[例5] マクロtestの呼び出し

```
test:Alt+I
```

PWB自体の機能拡張

作成から登録まで

PWBでは，ユーザー独自の拡張機能をCのプログラムを書いて，つけ加えることができます.

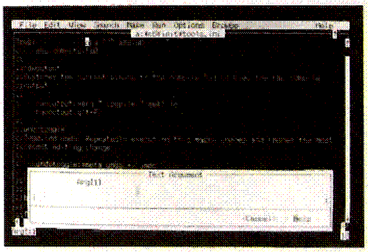


写真2　[Alt]+[A]操作をしてダイアログボックスを開く

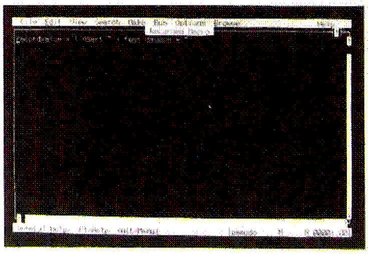


写真3　Edit Macroで記録したマクロを確認

ユーザーの作ったプログラムは，PWB起動時にPWBにリンクされます．ファイルの拡張子は，.MXT(MS-DOSの場合)，.PXTまたは.DLL（OS/2の場合）となります．感覚的にはDLLと同じなのですが，いったんPWBをたちあげてしまうとリンクを解除することはできません．

また，MS-DOSとOS/2のファイルをバインドさせることはできません．ただし，同じ機能をそれぞれのモードでコンパイルし，拡張子を除くファイル名を同一のものにすることで読み分けることができます．

このほかにも，MS-DOSの場合とOS/2の場合ではいくつか違いがあります．たとえば，作成する拡張機能のプログラムには，一部のCランタイム関数を使用することができるのですが，MS-DOSではこれにも制限があります（詳しくはマニュアルを参照してください）．OS/2では，EXTHDRPD.OBJおよびDLLランタイムライブラリとリンクすることによって，Cのランタイム関数がどれでも使用できるようになります．ただし，このようにして作成したプログラムでは，MS-DOSモードとの互換性はなくなります．

では，Cでの拡張機能の作成手順を見てみましょう．ここでは，拡張機能のソースファイル名をptes2.c，作成するユーザー関数名をMoveLineとして説明します．

①EXT.Hをインクルードする

②必ず次のオブジェクトを含める

swiTable	スイッチを定義する構造体配列
cmdTable	関数を定義する構造体配列
WhenLoaded	PWBが拡張機能をロードするときに実行される初期化関数

③ユーザー関数を作る（詳細は後述）

④拡張モジュールをコンパイルする

基本的なコンパイルパラメータは次のようになります．

```
cl -c -Gs -ACw ptes2.c
```

拡張機能コンパイルの場合は次の条件が必要です．

- スタックチェックなし
- DS != SS
- デフォルトのデータタイプが必ずfarであること

⑤拡張モジュールをリンクする

基本的なリンクパラメータ（OS/2）は次のようになります．

```
link /NOI  exthdrp  ptes2,
ptes2.pxt,,,ext.def
```

⑥拡張モジュールの呼び出し用の記述をTOOLS.INIに追加する

作成した拡張モジュールをPWBに読み込むためにTOOLS.INIファイルに書き加えます．

［例6］拡張モジュールの呼び出し

```
[pwb]
 ;
load:$INIT:ptes2
```

$INITは，環境変数INITの指すディレクトリにファイルが存在することを意味しています．INIT以外の環境変数を使ってもかまいません．ただし，$の後ろに書く環境変数名は必ず大文字にします．

⑦ユーザー関数の呼び出し

作成したユーザー関数をPWBから呼び出すにはいくつかの方法があります．

TOOLS.INIを直接編集する

ユーザー関数をキー操作によって呼び出す場合は，さらにその定義をTOOLS.INIに追加する必要があります．例6の後ろに次の形式で1行つけ加えます．

ユーザー関数名：キーの名前

「Options」メニューの「Key Assignments...」で変更する

キー操作を登録して，「Options」メニューから「Key Assignments...」を選択すると，新たに次の項目が表示されます．

例7を例8のように書き換えれば，TOOLS.INIを直接変更しなくても，キーを割り当てることができます．

［例7］キー操作が未定義の場合

```
[pwb-ptes2]
; MoveLine:is unassigned
```

［例8］キー操作をTOOLS.INIに定義している場合

```
[pwb-ptes2]
MoveLine:Alt+C
```

ユーザー関数呼び出しマクロを作成する

新たにマクロを作成してそこからユーザー関数を呼び出す方法です．マクロの実行のところでも説明しましたが，マクロに割り当てられたキーの定義は「ASSIGN」という疑似ファイルで参照することができます．疑似ファイル「ASSGIN」を見ると，unassignedに

統合開発環境用語集
条件コンパイル　ヘッダ指令を使用して，コンパイルする部分や，しない部分を作ること．

なっているマクロや関数の前には，「；」マークがついています．疑似ファイルでは「；」ではじまる行はコメントとみなされます．キー割り当てを登録するときには，このマークを消してください．疑似ファイルを書き換えて保存すると，TOOLS.INIが変更されて［pwb］セクションの前に追加されます．

［例 9］

```
[pwb-ptes2]
MoveLine : Alt+C
```

この機能は，デフォルトでユーザーが定義した拡張機能関数のキー割り当てを，さらにカスタマイズしたときに必要です．というのは，デフォルトのキー定義をTOOLS.INIの［PWB］セクションに書き加えた場合，その定義は，「Options」メニューからは変更できず，新しいキー定義がTOOLS.INIに追加されるだけだからです．これを利用すれば，デフォルトのキー定義を残したまま，ほかのキーを拡張関数に割り当てることができます．

ユーザー関数にキー割り当てを埋め込む

ユーザー関数を作成するときに，SetKeyという特殊な関数を使えば，プログラムの中であらかじめ呼び出し用のキーの割り当てを行うこともできます．この方法で作成したプログラムを使用する場合，TOOLS.INIファイルにキー定義を書き加える必要はありません．

Execute関数を呼び出す

キーによる呼び出し以外に，マクロのときのようにExecute関数を呼び出して実行することも可能です．

PWB内から拡張機能をビルドする場合は，前述の④以降の手順は次のようになります．

④「MAKE」メニューから「Set Program List...」を選択して，EXT.DEF及びPTES2.Cを登録する
⑤「Options」メニューの「Build Options...」から「OS/2 Extention」を選択する
⑥デフォルトでOS/2ライブラリを選択している場合は，「C Compile Options...」で「Default Libraries」を選択する
⑦「Make」メニューから「Build」を開始する
⑧TOOLS.INIファイルに，次の1行を加える

```
load:$INIT:PTES2
```

このとき，ソースリストの中で拡張機能を呼び出すキー定義をしていない場合は，キーを割り当てる
⑨TOOLS.INIを保存してPWBを再度たちあげる

拡張機能プログラミングの実際

サンプルptes2.cを例にして

では，サンプルプログラムptes2.cを使って，実際にPWBに拡張機能をつけ加える例を見ていきましょう．サンプルptes2.cでは，ユーザー関数MoveLineという機能をもっています．機能MoveLineは，カーソルを指定した行数だけ，後ろまたは前に移動します．

拡張機能を呼び出すには，PWBに対して機能の名前とアドレス，引数の型，使用するスイッチなどを，配列cmdTableおよびswiTableによって与える必要があります．

◆ext.h

拡張機能用のヘッダファイルです．後述する拡張機能用のcmdDesc型などは，このヘッダファイルに定義されています．拡張機能プログラムを作成するときは，必ずインクルードしてください．

◆PWBFUNC

ユーザー関数をPWBから呼び出すためには，必ず関数の型をPWBFUNCとし，関数のプロトタイプ宣言をします．

◆cmdTable

拡張機能プログラムは，必ずcmdTableという配列をもちます．PWBは拡張機能を呼び出すとき，必要な情報をこの配列から得ます．cmdTableは，cmdDesc型の構造体の配列です．

cmdDesc構造体のメンバは次の4つです．配列の最後の構造体は値NULLをとります．

機能の名前
関数のアドレス
予約項目（必ず0）
機能が受け取る引数の型

◆swiTable

swiTableは，PWBとユーザー関数との間でスイッチに関する情報を交換するための配列です．この配列に値をセットすると，拡張機能側からPWB

のスイッチを変更することができます．swiTableは，swiDesc型の構造体の配列です．swiTableのswiDesc構造体のメンバは次の3つです．配列の最後の構造体は値NULLをとります．

スイッチを差すポインタ
スイッチまたは関数を差すポインタ
スイッチの型を表すポインタ

拡張機能で使用するスイッチは，TOOLS.INIで設定することもできます．PWBは拡張機能をロードするときに，TOOLS.INIファイルの［pwb-ext］セクションを参照します．このセクションに

［pwb-拡張機能ファイルの名前］

を追加し，この項目の後ろにスイッチの設定を加えると，拡張機能中でこのスイッチの設定が有効になります．

◆WhenLoaded

PWBの拡張機能は，必ずWhenLoadedという関数をもっていなければなりません．WhenLoaded関数は，ユーザー機能が必要とする初期化を行います．ユーザー関数を呼び出すためのデフォルトのキー設定をしたい場合，ここでSetKey関数を用いて定義します．初期化の必要のないときにはこの関数は何もしません．

［例10］

```
void EXTERNAL WhenLoaded(void)
{
        DoMessage("Loading My PWB Function ");
}
```

◆ユーザー関数

ユーザー関数は，3つのパラメータをもちます（表3）．通常，拡張機能に

```
PROJ = ptes2
PROJFILE = ptes2.mak
DEBUG = 0

PWBRMAKE  = pwbrmake
NMAKEBSC1  = set
NMAKEBSC2  = nmake
CC  = cl
CFLAGS_G  = /Asfu /W2 /G2 /Zp /BATCH
CFLAGS_D  = /qc /Gs /Zi /Od
CFLAGS_R  = /Ot /Oi /Ol /Oe /Og /Gs
MAPFILE_D  = NUL
MAPFILE_R  = NUL
LFLAGS_G  = /NOI /BATCH
LFLAGS_D  = /CO /FAR /PACKC /PACKD /PMTYPE:VIO
LFLAGS_R  = /EXE /FAR /PACKC /PACKD /PMTYPE:VIO
RT_OBJS  = exthdrp.obj
LINKER  = link
ILINK  = ilink
LRF  = echo > NUL

DEF_FILE  = EXT.DEF
OBJS  = PTES2.obj
SBRS  = PTES2.sbr

all: $(PROJ).pxt

.SUFFIXES:
.SUFFIXES: .sbr .obj .c

PTES2.obj : PTES2.C

PTES2.sbr : PTES2.C


$(PROJ).bsc : $(SBRS)
        $(PWBRMAKE) @<<
$(BRFLAGS) $(SBRS)
<<

$(PROJ).pxt : $(DEF_FILE) $(OBJS)
!IF $(DEBUG)
        $(LRF) @<<$(PROJ).lrf
$(RT_OBJS: = +^
) $(OBJS: = +^
)
$@
$(MAPFILE_D)
$(LLIBS_G: = +^
) +
$(LLIBS_D: = +^
) +
$(LIBS: = +^
)
$(DEF_FILE) $(LFLAGS_G) $(LFLAGS_D);
<<
!ELSE
        $(LRF) @<<$(PROJ).lrf
$(RT_OBJS: = +^
) $(OBJS: = +^
)
$@
$(MAPFILE_R)
$(LLIBS_G: = +^
) +
$(LLIBS_R: = +^
) +
$(LIBS: = +^
)
$(DEF_FILE) $(LFLAGS_G) $(LFLAGS_R);
<<
!ENDIF
        $(LINKER) @$(PROJ).lrf


.c.sbr :
!IF $(DEBUG)
        $(CC) /Zs $(CFLAGS_G) $(CFLAGS_D) /FR$@ $<
!ELSE
        $(CC) /Zs $(CFLAGS_G) $(CFLAGS_R) /FR$@ $<
!ENDIF

.c.obj :
!IF $(DEBUG)
        $(CC) /c $(CFLAGS_G) $(CFLAGS_D) /Fo$@ $<
!ELSE
        $(CC) /c $(CFLAGS_G) $(CFLAGS_R) /Fo$@ $<
!ENDIF


run: $(PROJ).pxt


debug: $(PROJ).pxt
```


統合開発環境用語集
スクロールバー　ウィンドウの右端や下端に表示される．このバーをマウスで操作して上下左右に画面をスクロールさせることができる．

リスト2
ptes2.mak

```
/* PWB custamize  Sample program MoveLine */

#include <ext.h>

PWBFUNC MoveLine( unsigned argData,ARG _far *pArg,flagType fMeta );

struct swiDesc swiTable[] =
{
        { NULL, NULL, 0 }
};

struct cmdDesc cmdTable[] =
{
        { "MoveLine", MoveLine, 0, NOARG },
        { NULL, NULL, 0, 0 }
};

void EXTERNAL WhenLoaded( void )
{
        SetKey("MoveLine","Alt+P");
        DoMessage("Loading PTest extension");
}

PWBFUNC MoveLine( unsigned argData,ARG _far *pArg,flagType fMeta )
{
        PFILE pFile;
        LINE  yPoint,yOffset,yEnd;
    COL   xPoint;
        char CursorPoint[5],*mi;
        int yn,MinusFlag;

        pFile = FileNameToHandle( "", "" );
        GetCursor(&xPoint,&yPoint);

        if (GetString(CursorPoint,"カーソルを移動させるライン数を入れてください"
                ,FALSE))
                return FALSE;
        mi = CursorPoint;

        if (CursorPoint[0] == '-'){
                mi++;
                if (!(yOffset=atou(mi))){
                        DoMessageBox("指定された数値は無効です。",NULL,NULL,MBOX_OK,0);
                        return FALSE;
                }
                yPoint -= yOffset;
        }
        else {
                if (!(yOffset=atou(mi))){
                        DoMessageBox("指定された数値は無効です。",NULL,NULL,MBOX_OK,0);
                        return FALSE;
                }
                yPoint += yOffset;
                yEnd = FileLength(pFile);

                if (yPoint>yEnd){
                        yn = DoMessageBox("移動先のライン数がEOFを越えました。",
                                        "ファイルの最後にカーソルを移動させますか?"
                                        ,NULL,MBOX_YESNO,0);

                        if (yn = MBOX_IDYES)
                                yPoint = yEnd-1;
                        else
                                return FALSE;
                }
        }

        MoveCur(xPoint,yPoint);
        return TRUE;
}
```

パラメータ	意味
argData	ユーザー関数を呼び出す時に使われるキー
pArg	ユーザー機能に渡す引数を含む構造体を指すポインタ
fMeta	metaが引数の前にあればTRUE (!=0) そうでなければFALSE (=0)

表3
ユーザー関数のパラメータ

おいて第1パラメータが使われることはほとんどありません．第2パラメータは，ARG型構造体へのポインタです．ARG型構造体は，表4のようなメンバをもっています．

ユーザー関数では，pArg->argTypeで，PWBがどの型のパラメータを渡したかを判別し，型に応じた処理を行います．PWBから渡される引数の型についての詳細は，PWBの「Help」の「CommandTableFlags」を参照してください．PWBからユーザー関数が受け取る引数の型は，cmdTable構造体の中で指定します．プログラムの構造は，PMやWindowsにおけるメッセージ処理のルーチンによく似たものとなります．

また，リアルモードではCの標準ライブラリのルーチンのうち，次のものは使用することができません．

- Cのスタートアップサポートが必要な関数
- メモリ管理ルーチン (mallocなど) と，これらを呼び出すルーチン
- プロセス制御ルーチン

◆return

最後に呼び出されたPWBに対して，正常終了したことを知らせるためにTRUEを返します．何らかの理由でエラーとなって，処理を中断してPWBにもどるときは，FALSEを返します．

◆Movelineの実行

ユーザーが［ALT］+［T］を入力すると，ダイアログボックスが開かれます．ここで，現在いる位置から何行後，前（数字の前に「－」をつけます）にカーソルを進めるかを入力します．まちがった入力をしたり，入力した行数が最大行数をこえたときには，ダイア

メンバ	意味
argType	引数の型を示す整数
arg	共用体

表4 ARG型構造体のメンバ

ログボックスを開いてメッセージを表示します．

ユーザーからの入力をまかなう関数は，エラーがあったかどうかを必ずプログラム側に返すようになっていますので，なるべくIFなどを使って処理したほうがよいでしょう．

DOS環境では，何かと批判のあるPWBですがOS/2で使ってみると，残りメモリの量を気にしなくてもいいこと，さまざまな機能を追加できる拡張機能が自由に使えるなど使いごこちは悪くありません．最近は，Windows 3.0の登場でやや影のうすくなった感のあるOS/2ですが，プログラム開発には非常に便利だといえます．

現在のPWBは，複雑なことができる反面，マニュアルの不備や細かすぎる設定のために，ユーザーが混乱を招くケースが多数見られます．次回のバージョンでは，このあたりが改善されることを期待したいと思います．

■PWB関数

Arg	関数または関数への引数を導入する
Assign	マクロを定義し，スイッチを設定する
Backtab	カーソルを1つ前のタブ位置へ移動する
Begfile	カーソルをファイルの先頭へ移動する
Begline	カーソルを行頭へ移動する
Cancel	現在の処理をキャンセルする
Cdelete	改行文字以外の直前の文字を削除する
Compile	コンパイルまたはビルドコマンドを実行する
Copy	カレント行をクリップボードにコピーする
Curdate	現在の日付を挿入する
Curday	在の曜日を挿入する
Curfile	カレントファイル名を挿入する
Curfileext	カレントファイルの拡張子を挿入する
Curfilenam	カレントファイルのベース名を挿入する
Curtime	現在の時刻を挿入する
Delete	改行文字以外のカーソル位置の文字を削除する
Down	カーソルを1行下へ移動する
Emacsdel	改行文字を含め，直前の文字を削除する
Emacsnewl	カレント行を終了し，新しい行を開始する
Endfile	カーソルをファイルの末尾へ移動する
Endline	カーソルを行末文字の右へ移動する
Environment	環境設定を表示，または変更する
Execute	一連の編集関数またはマクロを実行する
Exit	PWBを終了する
Graphic	ファイルにキーのASCII値を挿入する
Home	カーソルをウィンドウの左上隅へ移動する
Information	以前に編集したファイルのリストを表示する
Initialize	初期化ファイルを再度読み込む
Insert	強調表示された領域に空白を挿入する
Insertmode	挿入モードのオン／オフを切り換える
Lastselect	最後のカーソル移動引数を再度呼び出す
Lasttext	最後に入力したtextargを再度呼び出す
Ldelete	カレント行を削除し，クリップボードに格納する
Left	カーソルを1文字分左へ移動する
Linsert	空白行を挿入する
Mark	カーソルをファイル内の指定の位置へ移動する
Maximize	ウィンドウを最大サイズにする
Message	ダイアログ行にメッセージを表示する
Mgrep	一連のファイルを検索する
Mlines	ウィンドウを行単位で後方へ移動する
Mpage	カーソルを1ページ，後方へ移動する
Mpara	カーソルをパラグラフ単位で，後方へ移動する
Mreplace	一連のファイル全体で置換する
Msearch	後方に検索
Mword	カーソルをワード単位で，後方へ移動する
Newline	カーソルを次の行へ移動する
Nextmsg	カーソルを次のエラーメッセージへ移動する
Noedit	編集禁止状態を切り換える
Paste	クリップボードからテキストを挿入する
Pbal	括弧や角括弧が左右そろっているかどうかをチェックする
Plines	ウィンドウを行単位で前方へスクロールする
Ppage	カーソルを1ページ，前方へ移動する
Ppara	次のパラグラフの先頭行にカーソルを移動する
Print	ファイルの一部または全部を出力する
Prompt	マクロにプロンプトを表示させる
Psearch	前方に検索
Pwbhelp	ヘルプを表示する
Pwbhelpnext	ヘルプの次のトピックを表示する
Pword	カーソルをワード単位で，前方へ移動する
Qreplace	確認つきで置換する
Quote	次のキー入力を，その文字自体として処理する
Record	マクロの記録のオン／オフを切り換える
Reflesh	編集内容を破棄して，ファイルを再度読み込む
Repeat	直前のコマンドを繰り返す
Replace	確認なしで置換する
Resize	アクティブウィンドウを拡大，縮小する
Restcur	Savecurで保存したカーソル位置を復元する
Right	カーソルを1文字分右へ移動する
Saveall	変更したファイルをすべて保存する
Savecur	カーソル位置を保存する
Sdelete	改行文字を含め，テキストを削除する
Searchall	一致した文字列すべてを強調表示する
Setfile	カレントファイルの直前に編集したファイルに切り換える
Sethelp	1つのヘルプファイルを追加または削除する
Setwindow	ウィンドウを再表示する
Shell	シェルを実行する
Sinsert	空白を挿入し，必要に応じて改行する
Tab	カーソルを次のタブ位置まで移動する
Tell	キー割り当て，マクロ定義を表示する
Undo	直前の編集関数，マクロを取り消す
Up	カーソルを1行上へ移動する
Usercmd	PWBのRunメニューに追加したコマンドを実行する
Window	ウィンドウを作成，消去，または移動する
Winstyle	ウィンドウスタイルを切り換える

統合開発環境用語集
ステータスバー　画面の一番下などで，現在の状態や，ハイライトとなっているコマンドの説明などを表示する．

計画的に使うことで効率を高める

MS-DOSの開発環境／QuickCの使い方

小松伸行

QuickC は Turbo C とならぶ代表的な統合環境 C コンパイラだが，MS-C の陰に隠れてか，初心者用というイメージが強いようだ．最初のバージョンでこそ，統合環境版コンパイラではミディアムモデルしか作成できないという制限があったが，Ver.2.0 では機能的に MS-C Ver.5.1 をしのぐものもある．また，Quick シリーズに共通する特徴として，強力なアドバイザ機能をもつ．MS-C の最新バージョン 6.0 が大量のディスクとメモリを必要とするようになった現在，MS-DOS 上での C コンパイラとしての QuickC を，もう一度評価すべき時期がきているのではないだろうか．ここでは，QuickC の使い方と，小規模な開発でアドバイザ機能を有効に利用するユニークなアイデアを紹介する．

登場した当初こそMS-Cから機能を削り取って統合環境を取り入れた，入門者用の低価格コンパイラというイメージがあったQuickCですが，現在のVer.2.0はMS-DOS上のプログラム作成には十分な機能をもちながら，低価格を維持しているコストパフォーマンスの高いC言語処理系といえます．

ここでは，QuickCの統合環境と使い方をプログラムを開発する手順を通して説明するとともに，QuickCの特徴のひとつであるヘルプデータベースを利用したプログラムのドキュメント化のアイデアを紹介します．

QuickCとMS-C
どちらを使うか？

Microsoftには，MS-C，QuickCという２つのC言語処理系があります．MS-Cは，統合開発環境という点ではQuickCに遅れをとっていましたが，Ver.6.0になってPWB（Programmer's WorkBench）を備え，QuickCと遜色がないどころか，QuickCを越える統合環境をもったといえます．ならば，金銭的な問題がなければMS-Cを使えばいいのかというと，そうもいいきれません．２つの処理系は性格が異なっているのです．

MS-Cは，MS-DOS版のみでも12.5MB，OS/2版も含めてすべてをインストールすると約23MBとかなりのディスクを消費するうえ，コンパイル速度もあまり速いとは感じられません．そのかわり，OS/2やMS-Windows上で走るプログラムのコンパイルが可能で，最適化能力も優れています．QuickCは，各メモリモデル用のライブラリをすべてインストールして約6MB．ひとつのメモリモデル用ライブラリだけなら3MB弱のディスク消費量ですみます．必要なディスク容量は，MS-Cよりずっと小さくてすみます．コンパイル速度もはるかに軽快に感じます．

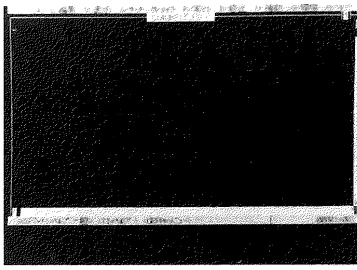
MS-Cの高機能を必要とする方は，MS-Cをどうぞご利用ください．でも，MS-DOS上で動かすプログラムを作成するならQuickCのほうがいいと思います．とくに，自分専用のプログラムを作るのなら，QuickCで決まりでしょう（無理をしてMS-Cの昔のバージョンのものを購入して，ずるずるとバージョンアップの泥沼にはまってしまった人を除けば）．

QuickCの統合環境
プログラム作成からデバッグまで

パーソナルコンピュータ上の統合開発環境という点では，QuickCはどちらかというと後発のものです．統合環境つきのコンパイラが世の中に出てからかなりの年月がたつわけで，それ自体めずらしいものではありません．代表的な統合環境コンパイラ製品は，ほとんど同じような外観と機能をもっています．とはいえ，ちょっとしたキーの割り当てやウィンドウの配色といった細かいことで，ユーザーが使いやすく思えるかどうかが大きく分かれてしまいます．そこがユーザーインターフェースの恐さでもあります．まずは，QuickCの統合環境全般についてみていくことにしましょう．

◆起動画面

起動時の画面は，最上部にメニュー行があり，最下行に簡単なガイド表示やカーソル位置などのステータス表示



画面１　QuickCの起動画面

が行われます．その２行を除いた部分には，編集用やヘルプ用のさまざまなウィンドウが必要に応じて開かれます．

◆エディタ

プログラマがもっとも頻繁に使う機能といえば，やはりエディタでしょう．QuickCのエディタには，ふつうに使う分には不満がない程度の基本的な機能は揃っています．さすがに，エディタとして市販されているものと比べると見劣りがしますが，統合環境があまり複雑で大きなものになったら，ユーザーの利用できるメモリが足りなくなってしまいますし，全体のバランスからいってがまんできる範囲のものです．

私が以前に使っていたあるコンパイラの統合環境版では，一時に１つのファイルしか編集することができませんでした．QuickCでは，ノートウィンドウを利用すれば同時に２つのソースを見比べることができます．ほぼ同時に参照，編集可能なファイルは，４つの通常ファイル＋ヘルプウィンドウの５つです．おもに編集に使うのはソースウィンドウです．ソースウィンドウでは「GRPH/F/F/再読込（f･2)」を使

統合開発環境用語集
ソースレベルデバッガ　ソースファイルを参照しながら，プログラムの実行ができるデバッガ

って，２つのファイルを交互に切り換えて編集できます．また，補助的にノートウィンドウを使うことができます．ノートウィンドウでも同じように２つのファイルを交互に編集可能ですから，合計４種類のファイルを編集できるわけです．このほかにヘルプ画面を参照できます．５つのファイルの間では相互にCut & Pasteができますから（ただしヘルプウィンドウへは書き込めません），この点では専用のエディタと遜色はないと思います．

編集用のコマンドは，カーソルキーとコントロールキーを押しながら別のキーを押すスタイルです．「GRPH/E/編集」メニューにはおもにCut & Pasteの機能が，「GRPH/S/サーチ」メニューには検索系のコマンドが含まれています．どちらもおもだったものは，ショートカットキーといって，ワンタッチで，呼び出すことも可能です．また，編集コマンドのキー割り当てを変更するツールもついています．

ただし，QuickCのエディタには基本的な編集機能しかありませんから，正規表現で検索しようとか，複雑な編集マクロを使いたいとか，２つのソースファイルの相違を調べたいとか，そういったことはできません．ただし「GRPH/U/C/自作メニュー...」で，QuickCからほかのエディタやツール類を呼び出せるよう設定できます．編集中のファイル名などのマクロ名も用意されていますから，外部エディタに編集中のファイルを自動的に読み込ませることも可能です．外部のツールでたりないところを補って使うのが賢い使い方でしょう．

◆コンパイル

エディタでプログラムを書いたら，コンパイル，リンク，実行ということになります．これには，[f･5]キーを押すだけで十分です．コンパイル中は，途中経過が画面に表示されます．コンパイル中にエラーが見つかると，コンパイルが終わってからエラーウィンドウが開かれて，修正の準備がされています．コンパイル，リンクでエラーがなければ，そのままできあがったプログラムの実行に移ります．

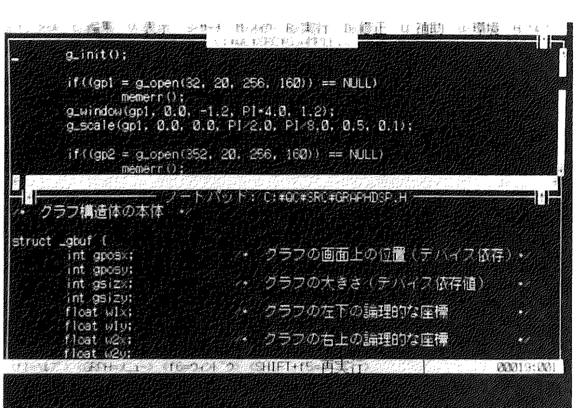
コンパイルに関連するメニューには，「GRPH/M/メイク」と「GRPH/R/実行」があります．EXEファイルを作るだけで，すぐには実行させたくないときは，「GRPH/M/B/プログラムファイル」を選択します．タイプミスのような単純なコンパイルエラーがないか確かめるだけなら，「GRPH/M/C/コンパイルファイル」でリンクせず，コンパイルするだけという指定もできます．コンパイル時のオプションを変更したときは，必ず「GRPH/M/R/再コンパイル」します．「プログラムファイル」と「再コンパイル」では，後述する分割コンパイルを行うときの動作が異なります．「GRPH/R/実行」メニューには，ステップ実行などのデバッグ機能も含まれています．

◆デバッグ

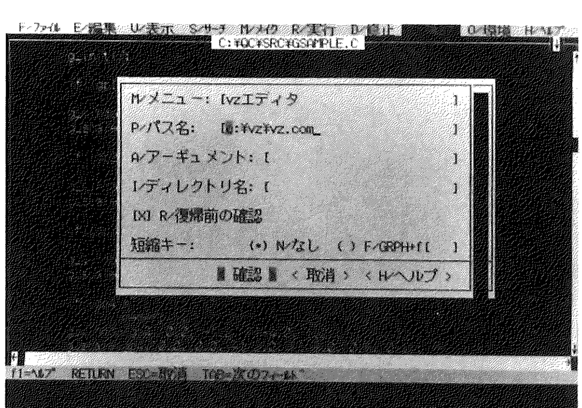
デバッグ関連のメニューは，おもに「GRPH/R/実行」と「GRPH/D/修正」メニューの下に集まっています．「GRPH/R/T/トレース（f･8）」と「GRPH/R/S/ステップ（f･10）」ではソースプログラムを１ステップずつ実行させます．「トレース」は呼び出している関数の中も１ステップずつ実行するのに対して，「ステップ」は関数の内部までトレースしません．

「GRPH/D/B/ブレークポイント...（f･9）」では，複数のブレークポイントを設定できます．実行中にブレークポイントに設定した行にさしかかると，プログラムはいったん実行を中断します．

「GRPH/D/P/ウオッチポイント...」を使うと，指定した変数が特定の値に変化したときにプログラムを中



画面２　複数のファイルを画面に表示する



画面３　自作メニューにコマンドを登録する

統合開発環境用語集
ダイアログボックス　ウィンドウの一種で，おもにユーザーからの入力を受け付けるのに使われる．

断できます.

「GRPH/D/W/ウオッチ変数...」では，専用のウィンドウに変数の値を表示しておけます．ただし,「GRPH/V/表示」メニューからローカル変数ウィンドウを開いておけば，現在実行中の関数に含まれるローカル変数の内容を自動的に表示してくれますから，グローバル変数を多用しなければ，いちいち「ウォッチ変数」指定を行う必要はありません.

デバッグ中に変数の内容を変更したいときは「GRPH/D/M/変数修正...」で可能ですが，ポインタが関係していたり変数の型が単純なものでなかったりすると，このコマンドは使えないのが残念です.

「GRPH/D/C/関数...」では，現在実行中の関数を呼び出した関数は何であるかmain()にさかのぼってすべてを引数つきで表示してくれます.

「GRPH/D/H/ヒストリ」では，デバッグ操作の記録をとっておくことができます.「ヒストリ」をonにしておくと,「GRPH/D/U/アンドゥ」で直前のコマンドを取り消したり（トレースを少し前に戻ってやりなおせる),「GRPH/D/R/リプレイ」で最初からデバッグ操作を再現したりできます．とくに，QuickC上でわずかな修正を加えただけならば，コンパイルしなおしても「リプレイ」可能ですから，きちんとデバッグできているかどうか，すぐに確かめられます．また，デバッグ対象のプログラムがキー入力を要求している場合,「ヒストリ」記録中に同時に記録したキー入力を利用するかどうかを「GRPH/D/T/インプット省略」で選択できます.「インプット省略」すれば，同じデバッグ操作を行いながら，キー入力だけは毎回変えて，その影響を観察することもできます.

ひととおり見ると，QuickCのデバッグ機能を使えばプログラムのすべての動きが確かめられるようにすら思えます．しかし，それで本来のデバッグが本当に楽になるかといえば，なかなかそうも思えません．経験を積んだプログラマなら，プログラムの動作に問題が生じたときにすぐにどのモジュールが悪いのか見当がついて，あまりデバッグ機能のお世話にならないはずです．いきなりステップ実行するような，いきあたりばったりのデバッグ方法では，結局は何もわかりません.

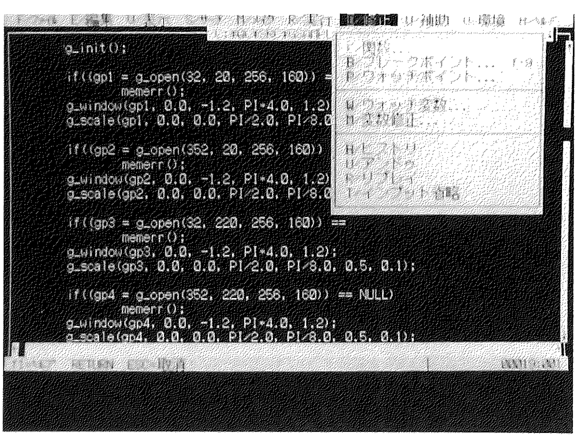
バグらしきものが見つかったら，入力を変えたりして症状をよく観察し，どこに問題があるのか絞り込む．次に問題のありそうなモジュールのソースをじっくり読んで，それでもどこが悪いか納得がいかなければ，はじめてトレースして自分の勘違いを確かめる．こういうときに,「ブレークポイント」や「ウオッチポイント」の機能を使えば効果的です.

QuickCのデバッグ機能がすぐれているといっても，静的なアルゴリズム実現上のミスを見つけるのがせいぜいで，割り込みがらみのバグにはまったくお手上げです．それに統合環境はメモリ食いですから，デバッグ時に配列の大きさを小さくしたりするとメモリがらみのバグが再現しなくなったりします．デバッグ機能の非力さ，利用上の手間，などを考えれば，プログラム全体に対してきちんとした設計をして，モジュールをきちんと下から積み上げていくプログラミングスタイルをとること．それに勝るデバッグ手法はないと思います.

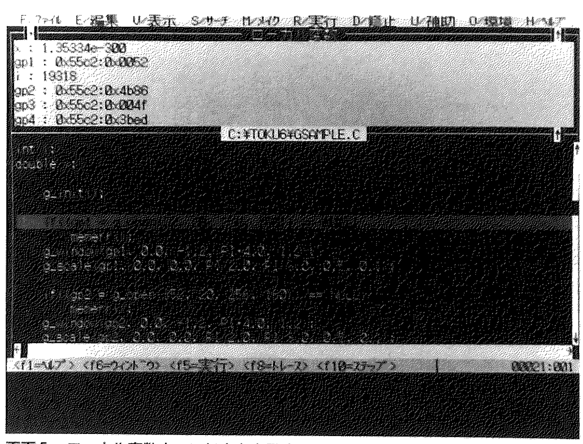
QuickC アドバイザ

マニュアルは不要

QuickCでは，[f･1]キーでヘルプを



画面 4　「GRPH/D/修正」メニュー



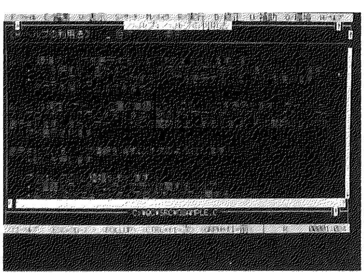
画面 5　ローカル変数ウィンドウを表示する

統合開発環境用語集
タイトルバー　ウィンドウの上部にあり，そのウィンドウの名称などを表示する．ここでマウスカーソルをドラッグするとウィンドウを移動できる.

呼び出せます．関数のリファレンスは当然ですが，いろいろなメニューの機能を調べるのも，エラーコードの意味を確かめるのもすべて［f･1］キーです．基本的に，カーソルがどこにあるかで表示されるヘルプの項目が決められます．どの関数が適当かわからなければ直接調べることはできませんが，ヘルプメニューの索引か，ヘッダファイルから探すのがよいでしょう．マイクロソフト社ではヘルプ機能とは呼ばずに，「QuickCアドバイザ」としています．

しかし，その分マニュアルが薄くなっています．喜ぶべきか悲しむべきか（私は悲しんでいるのですが）関数リファレンスなどは，手帳みたいなものが入っているだけ．C言語入門用のガイドと統合環境版でないコンパイラなどのマニュアルは厚いのですが，統合環境版のマニュアルらしきものは，“Up and Running”というタイトルの100ページに満たないもので，インストール方法の説明の後ろに簡単な環境の解説をしてQuickCアドバイザの使い方が書いてあるだけ！　あとはすべて実際に使いながらヘルプ画面を読んでくださいという態度です（おっと，忘れていました．CBTという統合環境のオンライン学習用ソフトもついています．最初に動かしたときは感動しました）．

それだけこのQuickCアドバイザに自信があるのでしょうが，とにかくはじめのうちはあらゆる場所でQuickCアドバイザを呼び出してみてください．私の知り合いのあるMacユーザーは，「ソフトを買ってインストールしたら，マニュアルは片づけてとにかくあちこちマウスでクリックするんだ」といっていましたが，QuickCにも同じことがいえます．［f･1］キーを押しまくってください．そうやって慣れていくのが統合環境に適応するこつだと思います．



画面6　QuickCアドバイザ

統合環境というものは，ウィンドウが何種類使えるとか，デバッグがしやすいとかいう点で評価されやすいと感じるのですが，QuickCの売りはこのQuickCアドバイザだと思います．さらに，QuickCアドバイザで呼び出すファイルを自作するためのユーティリテイが付属していますので，自分でヘルプファイルが作れます．あとでその利用法を紹介します．

Make機能

分割コンパイルには必須

Cでは，ある程度複雑なプログラムは複数のファイルに分割して作成します．分割コンパイル&リンクなどと呼ばれている技法です．複数のファイルに分けることにより，共通に使える関数群の部品化が行えるうえ，プログラムの一部を手直しする場合に，直したファイルだけコンパイルすればすむので，コンパイルの時間が節約できるというメリットがあります（もっともQuickCではコンパイルにあまり時間が必要ではありませんが）．

ここでは例として，少し大きなものを用意しました．このプログラムは，折れ線グラフを画面表示してみるものです．もとは計測用のプログラムの一部で，いくつかの測定器から送られてくるデータを常にモニタするために使っていた部分を簡単にしたものです．QuickCには，グラフ表示をする関数群もついているのですが，複数のグラフを画面に表示しながら少しずつデータを付け加えていくという用途には使えなかったため，別途作成しました．gsample.cがメインプログラムで，graphdsp.cとgraphdsp.hがグラフ表示用のモジュールになっています．

これらのファイルをコンパイル，リンクする指定を行うには,「M/メイク」メニューの「S/プログラムリストの設定」を使います．プログラムリストとは聞き慣れない言葉ですが，いわゆるmakeファイルのことです．makeファイルには，ソースファイルの依存関係を記述しますが，さらに効果的に使うには，上手なマクロの利用法などの知識やある程度の経験が必要です．QuickCでは，makeファイルを書き下さなくても，メニューからどのプログラムをリンクすればいいかを選ぶだけで，makeファイルを組み立てていくことができます．

では，実際にやってみましょう．「M/S/プログラムリストの設定」を選ぶと，どのファイルをプログラムファイルとするか設定するウィンドウが開きますから“GSAMPLE.MAK”に指定します．“GSAMPLE.MAK”というファイルが存在しなければ，プログラムファイルを最初から編集する画面に変わります．makeファイルの名前は通常は“makefile”を使うことになっていますが，QuickCではメインプログラムのファイル名の拡張子を“.MAK”にしたものを使います．こうし

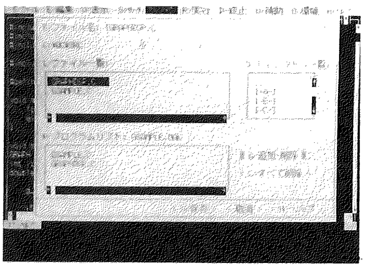
統合開発環境用語集
タイリング　複数のウィンドウを，それぞれが重ならないように表示させること．

ておくと，

A>QC gsample.c

などとしてQuickCを立ちあげたときに「GSAMPLE.MAK」を利用するかどうかを確認してきます．

次に，真ん中のウィンドウの「ファイル一覧」の部分にカーソルを移動して，必要なファイルの上でリターンキーを押せば，下の「プログラムリスト」ウィンドウに登録されていきます．誤って関係ないソースファイルを登録してしまった場合は，もう一度リターンキーを押せば取り消されます．



画面7　M/S/プログラムリストの設定

ただし，この方法で組み立てたmakeファイルは，1つのプログラムがどのファイルから構成されているかを指定するだけで，ファイル間の依存関係を示すというmakeファイルの重要な機能を利用していません．確かに「プログラムリスト」という名前どおり，必要なファイルをリストアップするだけです．でもこのファイルは特殊な形式ではなく，通常のテキストファイルですから，エディタで読み込んで書き換えることができます．今回の例では，gsample.objとgraphdsp.objではじまる行の最後に“graphdsp.h”を書き加えるだけです．それでも最初からmakeファイルを自分で書くよりはずっと楽なのは確かです．また，このmakeファイルは，付属のNMAKE.EXEの入力ファイルとして使えるようになっています．QuickCには，コマンドライン版のコンパイラも含まれていますから，

A>NMAKE -f gsample.mak

とすれば，統合環境を起動することなくプログラムのコンパイルとリンクが行われます．

リスト1
GSAMPLE.MAK

```
PROJ     =GSAMPLE
DEBUG    =1
CC       =qcl
CFLAGS_G        =
CFLAGS_D        =/Od /Zi /Zr
CFLAGS_R        =/O /DNDEBUG
CFLAGS  =$(CFLAGS_G) $(CFLAGS_D)
LFLAGS_G        =/NOI
LFLAGS_D        =/CO
LFLAGS_R        =
LFLAGS  =$(LFLAGS_G) $(LFLAGS_D)
RUNFLAGS        =
OBJS_EXT =
LIBS_EXT =

all:    $(PROJ).exe

gsample.obj:    gsample.c graphdsp.h

graphdsp.obj:   graphdsp.c graphdsp.h

$(PROJ).exe:    gsample.obj graphdsp.obj $(OBJS_EXT)
        echo >NUL @<<$(PROJ).crf
gsample.obj +
graphdsp.obj +
$(OBJS_EXT)
$(PROJ).exe

$(LIBS_EXT);
<<
        ilink -a -e "link $(LFLAGS) @$(PROJ).crf" $(PROJ)

run: $(PROJ).exe
        $(PROJ) $(RUNFLAGS)


```

ヘルプファイルを作ってみる

作成した関数をヘルプで参照

いったん作成した関数群をあとで再利用するには，ドキュメント類の整備が必要です．いつまでも内容を詳細に覚えているわけにはいきませんし，周りの人にも使ってもらおうと思うなら，関数の使い方のマニュアルを書いておくべきでしょう．今回例題としてあげたプログラムの主要部分も一種のグラフ表示ライブラリとして，あとあと使えそうな関数で構成されています．

マニュアルを書いてワープロできれいに清書すれば，印刷して使うのにはいいかもしれません．テキストファイルとして保存しておけば，プログラム作成中にエディタの中から読めて便利でしょう．でも，QuickCを使っているのなら，もう1つ別の選択ができます．

マニュアルをQuickCのヘルプファイルの形式にしてしまうのです．そうすれば，あとでこのモジュールを利用してプログラムを作ろうとしたときに，最初から提供されているライブラリ関

統合開発環境用語集
ダブルクリック　マウスのボタンを2回クリックすること．この動作によってすばやく目的のコマンドなどが選択できる．

数と同じ形でQuickCアドバイザを呼び出して読むことができます．簡単なサンプルプログラムを書いて，ヘルプファイルに登録しておけばCut ＆ Pasteして使うこともできます．実際に使いたいときに検索の手間がいらないのですから，単なるテキストファイルよりも具合がよさそうです．

◆ヘルプファイルの形式

QuickCアドバイザで直接参照できる形式のファイルをヘルプデータベースと呼びます．これは特別な形式に変換されていて，普通に読み書きすることはできません．この形式のファイルを作成,解読するのが,MAKEHELP.EXEという付属のユーティリティプログラムです．HELPMAKE.EXEへ入力するヘルプデータベースのもとになるテキストファイルを，ヘルプテキストファイルと呼びます．ヘルプテキストファイルの形式には3種類（QuickHelp形式，Richテキスト形式，簡易書式化ASCII形式）ありますが，ここでは表現力が高く，特別なソフトの不要なQuickHelp形式を使って説明します．

QuickCアドバイザは，キーワードの場所にカーソルがあるとき，そのキーワードを検索するように働きます．このキーワードをコンテキストと呼びます．また，ヘルプウィンドウの中でも，適当な文字列（独立した1単語とは限らない）の上で［f･1］キーを押せば別の場所を呼び出すことができます．これをハイパーリンクという特別な言葉で呼びます．実はその場所に見えないキーワードが入れられているのです．さらに見栄えをよくするために表示色などを指定することが可能です．そのために特別なコマンドや制御文字が用意されています．

命令	動作
:ln	オープンされるヘルプウィンドウのサイズをn行にして表示する
:n text	ヘルプのタイトルとしてtextを表示する
:p	ページ区切り指定

表1 コロン制御文字コマンドの意味

表示指定フラグ	機能(IBM PC系)	機能(98)
¥a	クロスリファレンス用の複数の単語を連続のものとみなす（共通）	
¥b, ¥B	太字指定	白色指定
¥i, ¥I	イタリック指定	緑色指定
¥p, ¥P	通常表示指定	黄色指定
¥u, ¥U	アンダーライン	反転表示指定
¥v, ¥V	表示・非表示属性の指定（ハイパーリンクの参照コンテキスト指定）（共通）	
¥¥	"¥"を表示（共通）	

（IBM PCでは"¥"を"＼"に読み変える）

表2 表示指定フラグの意味

代表的なコマンドとしては，コンテキストを定義する".context"コマンドがあります．あとにコンテキストの単語が続き，このコマンドから次の".context"コマンドまでの文章がヘルプウィンドウに表示されるのです．他の制御コマンドなどは表1，2を参照してください．

◆ヘルプデータベースへの変換

このようなコマンドや表示指定を駆使して，できるだけQuickCに付属のヘルプファイルに似せたものが，GDSPHLP.TXTです．リスト2にはじめのほうを抜粋しておきます．一部，説明していないコンテキストの指定を使っているところがありますが，詳しくは「Tool Kitマニュアル」のHELPMAKEの項を参照してください．

このヘルプテキストファイルをGRAPHDSP.HLPというヘルプデータベース形式に変換するには，MS-DOSのプロンプトから次のように入力します（残念ながらこの部分は統合環境化されていません）．

```
A>helpmake /A: /V /E /O graphdsp.hlp gdsphlp.txt
```

これで自分用のヘルプデータベースができました．でもこのままではQuickCから呼び出すことはできません．QuickCから使えるようにするには，マニュアルには何通りかの方法が書いてあるのですが，いちばん柔軟なうえに失敗しても安全な方法として環境変数を利用することをお勧めします．HELPFILESという環境変数に，自分用のヘルプファイル名をセットするのです．“；”で区切れば，最大6つの自分用ヘルプファイルを利用できます．今回の例では次のようにします．

```
A>SET HELPFILES=
GRAPHDSP.HLP
```

◆ヘルプファイル作りの奥義

チームでプログラムを作っている，自分が作ったライブラリをほかの人も使うという状況なら，時間をかけてヘルプファイルを作成する意義があると思います．もっとも，ここで紹介した方法は見栄えはいいのですが，実に手間がかかります．私自身もめったにここまでヘルプファイルに凝ることはありません．私がQuickCアドバイザで自作の関数を検索したいのは，その関数もまだ作成途上でドキュメント製作はおろかデバッグも十分でないというような状況のほうが多いのです．いちばん必要なときに使えないのでは，ヘルプファイルを作成する意味がありません．そこで，さらに簡便なやり方，マニュアル用のファイルをわざわざ作成するのではなく，プログラム自身をヘルプファイルに仕立ててしまう方法を紹介しましょう．

QuickCに出会う前は，エディタに作成中のプログラムに使うファイルすべてを読み込んでおいて，ある関数の

リスト2 GDSPHLP.TXT一部のみ抜粋

```
.context g_open
:110
:n折れ線グラフ表示用関数
¥b《S/要約》¥p  ¥i《¥a¥pD/概要¥vg_open.des¥v¥i》¥p  ¥i《¥a¥pE/プログラム例¥vSING
RAPH.C¥v¥i》¥p
--------------------------------------------------------------------------------
¥bｲﾝｸﾙｰﾄﾞ:¥p     <graphdsp.h>

¥bﾌﾟﾛﾄﾀｲﾌﾟ:¥p    GRAPH *g_open(int posx, int posy, int sizx, int sizy);

¥b戻り値:¥p      新規に作成されたグラフ構造体へのポインタ,
            エラーの際はNULLポインタ

¥b参照:¥p        g_window, g_scale, g_addpoint, g_close, g_init, g_term
.context g_window
:111
:n折れ線グラフ表示用関数
¥b《S/要約》¥p  ¥i《¥a¥pD/概要¥vg_window.des¥v¥i》¥p  ¥i《¥a¥pE/プログラム例¥vSI
NGRAPH.C¥v¥i》¥p
--------------------------------------------------------------------------------
¥bｲﾝｸﾙｰﾄﾞ:¥p     <graphdsp.h>

¥bﾌﾟﾛﾄﾀｲﾌﾟ:¥p    void *g_window(GRAPH *gp, double x1, double y1,
                                double x2, double y2);

¥b戻り値:¥p      なし

¥b参照:¥p        g_open, g_scale, g_addpoint, g_close, g_init, g_term
.context g_scale
:112
:n折れ線グラフ表示用関数
¥b《S/要約》¥p  ¥i《¥a¥pD/概要¥vg_scale.des¥v¥i》¥p  ¥i《¥a¥pE/プログラム例¥vSIN
GRAPH.C¥v¥i》¥p
--------------------------------------------------------------------------------
¥bｲﾝｸﾙｰﾄﾞ:¥p     <graphdsp.h>

¥bﾌﾟﾛﾄﾀｲﾌﾟ:¥p    void *g_scale(GRAPH *gp, double ox, double oy,
                               double xs1, double xs1,
                               double xs2, double ys2);

¥b戻り値:¥p      なし

¥b参照:¥p        g_open, g_window, g_addpoint, g_close, g_init, g_term
.context g_addpoint
:110
:n折れ線グラフ表示用関数
¥b《S/要約》¥p  ¥i《¥a¥pD/概要¥vg_addpoint.des¥v¥i》¥p  ¥i《¥a¥pE/プログラム例¥v
SINGRAPH.C¥v¥i》¥p
--------------------------------------------------------------------------------
¥bｲﾝｸﾙｰﾄﾞ:¥p     <graphdsp.h>

¥bﾌﾟﾛﾄﾀｲﾌﾟ:¥p    void *g_addpoint(GRAPH *gp, double x, double y);

¥b戻り値:¥p      なし

¥b参照:¥p        g_open, g_window, g_scale, g_close, g_init, g_term
.context g_close
:110
:n折れ線グラフ表示用関数
¥b《S/要約》¥p  ¥i《¥a¥pD/概要¥vg_close.des¥v¥i》¥p  ¥i《¥a¥pE/プログラム例¥vSIN
GRAPH.C¥v¥i》¥p
--------------------------------------------------------------------------------
¥bｲﾝｸﾙｰﾄﾞ:¥p     <graphdsp.h>

¥bﾌﾟﾛﾄﾀｲﾌﾟ:¥p    void *g_close(GRAPH *gp);

¥b戻り値:¥p      なし

¥b参照:¥p        g_open, g_window, g_scale, g_addpoint, g_init, g_term
.context g_init

:110
:n折れ線グラフ表示用関数
¥b《S/要約》¥p  ¥i《¥a¥pD/概要¥vg_init.des¥v¥i》¥p  ¥i《¥a¥pE/プログラム例¥vSING
RAPH.C¥v¥i》¥p
--------------------------------------------------------------------------------
¥bｲﾝｸﾙｰﾄﾞ:¥p     <graphdsp.h>

¥bﾌﾟﾛﾄﾀｲﾌﾟ:¥p    void g_init(void);

¥b戻り値:¥p      なし

¥b参照:¥p        g_open, g_window, g_scale, g_addpoint, g_close, g_term
.context g_term
:110
:n折れ線グラフ表示用関数
¥b《S/要約》¥p  ¥i《¥a¥pD/概要¥vg_init.des¥v¥i》¥p  ¥i《¥a¥pE/プログラム例¥vSING
RAPH.C¥v¥i》¥p
--------------------------------------------------------------------------------
¥bｲﾝｸﾙｰﾄﾞ:¥p     <graphdsp.h>

¥bﾌﾟﾛﾄﾀｲﾌﾟ:¥p    void g_term(void);

¥b戻り値:¥p      なし

¥b参照:¥p        g_open, g_window, g_scale, g_addpoint, g_close, g_init
.context g_open.des
:112
:ng_open 関数
¥i《¥a¥pS/要約¥vg_open¥v¥i》¥p  ¥b《D/概要》¥p  ¥i《¥a¥pE/プログラム例¥vSINGRAPH.
C¥v¥i》¥p
```

統合開発環境用語集

チュートリアル　学習ソフトともいう．統合開発環境を理解するための練習を行うことができる．

仕様がわからなくなると，エディタの画面をその関数が含まれているソースファイルに切り換えて，すでに作成してあるその関数自体のソースを読んで使い方を思いだしたものです．QuickCを使うようになってあるときふと気がつきました．関数名から，その関数のソースコード（とそれに付随したコメント）が検索できて，読めるだけで十分なのだと．そのためには，ほかで使う可能性のある関数の前に“.context 関数名”という1行を挿入するだけですむのです．

Cのソースプログラムにそのまま追加すれば，文法エラーになってしまいますが，“.context 関数名”という行をコメントにしてしまえば平気です．HELPMAKEプログラムはCの文法を知りませんから，コメントの内部にあろうが外部にあろうが，行の先頭から“.context …”と書いてあればちゃんと処理してくれます．

graphdsp.cを見てください．このプログラムで外部から呼び出される可能性のある関数のコメントの先頭には，すでに“.context”指令が埋め込んであります．

```
A>HELPMAKE /E /O
graphdsp.hlp graphdsp.c
```

とすれば，目的のヘルプデータベースが作られます．この方法ですと，ソースファイルを変更してもヘルプファイルを作成するための特別な手間はいりません．ただ，ソースファイルを大きく変更したときに（とくに公開している関数の外部仕様を変更した場合），忘れずに上のヘルプデータベースを作成するコマンドを実行すればよいのです．

もちろん，コメント内で“.context”コマンド以外の表示指定フラグや制御文字を使えますから，表示するときに飾りたててもかまわないのですが，そこまで凝る必要もないでしょう．せいぜい表示ウィンドウの行数を指定する“:l”指令があればよいと思います．その場合，“/A:”オプションを忘れずにHELPMAKEにつけてください．

簡易書式化ASCII形式とRichテキスト形式

●簡易書式化ASCII形式

簡易書式化ASCII形式は，ヘルプファイルのキーワードにあたるコンテキストと，それに対する解説の2つの部分で構成される形式のファイルです．

簡易書式化ASCII形式を使えば，HELPMAKEを使わずにヘルプテキストファイルのままQuickCアドバイザが参照できます．一見使いやすそうなのですが，コンテキストコマンドに相当するヘルプ区切り記号“>>”をファイルの先頭に必ず入れるという規則があるので，Cのソースプログラムをそのままヘルプファイルにする方法は使えません．

でも，QuickCを一種の簡易アイデアプロセッサと考えれば，エディタとQuickCアドバイザの部分だけ利用してドキュメント作成に応用することも可能です．その場合には，簡易書式化ASCII形式を利用するとよいでしょう．

●Richテキスト形式(RTF)

米Microsoft社が，同社のワープロソフトMS-WORDとほかのソフトの文書の互換性を保つために考案したファイル形式です．したがって，RTF形式のヘルプファイルを作成するためには，この形式をサポートするワードプロセッサなどのソフトが必要ということになります．

まとめ
計画性のある開発を

QuickCは，MS-DOS用のC言語処理系としては十分な能力があります．そのうえ，統合環境がついて，高機能エディタの半額ぐらいで購入できるというコストパフォーマンスは驚くべきものです．メニューの日本語訳には，ご愛敬といった点も見受けられますが，機能が増えた分だけ自分なりの使い方にさまざまな工夫ができると思います．

ただし，忘れてはならないのは，統合環境といっても，プログラムの設計やデバッグの際には，従来と同じく計画性が必要だということです．いきあたりばったりでは，すぐに限界がきます．MS-DOS上で使う以上，QuickCではどうしてもメモリの制約を受けてしまいます．少し大きなデータをメモリ上に置く必要があったりすると，すぐに統合環境が使えない，使いにくくなってしまうというのが現在の状況です．

大きいプログラムを効率的に開発するには，統合環境とコマンドライン版のコンパイラをうまく使い分ける必要があります．全体をきちっと設計し，部品となる下請けの関数類は統合環境で開発し，十分にデバッグをしておく．全体を段階的に結合していくときにはコマンドライン版のコンパイラを利用するといった工夫が必要でしょう．

統合開発環境用語集
テキストボックス　コマンドの実行に必要な情報を入力するためのボックス．ダイアログボックス内にある．

INTERVIEW

マイクロソフトに聞く

プロ指向とQuickシリーズの2本立てで展開

マイクロソフト株式会社 第一研究開発部言語課主務

能祖裕樹

BASICでは元祖というプライドが

——統合環境の製品としては，Quickシリーズが最初ということになりますね．

能祖 統合環境商品のはしりといえるのはボーランドのTurbo Pascalです．当時，Borland Internationalは，Pascalの成功をもとに，それをC，BASICへと展開してマイクロソフトの言語製品のシェアをとるということを雑誌で発表しました．実際に，Turbo BASICという名前で，MicrosoftBASICと完全互換のBASICが，Turbo Pascalのような画面をもって，ファンクションキーでエディットでき，実行もできる製品になるというのです．そこで，BASICだけは先にださなければと，がんばって作った製品がQuickBASICです．いわゆる統合環境をもったものとしては，弊社ではQuickBASICが最初ということになります．

最初のQuickBASICは，1行入力するとすぐにバイナリコードに変換して，実行スピードを非常に高速にするインラインコンパイル機能，それと簡単なものでしたがトレースなどのデバッグ機能を実現していました．Turbo PascalやTurbo Cなど，ボーランドのプロダクツでは，当時はまだデバッグ機能をインクルードしていませんでした．「デバッグ」という名称の機能はありましたが，それは単純にコンパイルのエラーを表示するだけだったのです．

日本ではQuickBASIC 4.2が最初で，アメリカにはないバージョンです．アメリカの当時のQuickBASIC 4.0をもとに，日本語処理機能を付加して，QuickBASIC 4.2として出荷しました．

同時に進行していたQuickCも，アメリカでは1.0だったのですが，日本では若干モディファイして1.1というバージョンでだしました．

——こうした開発環境を何と呼んでいるのですか？

能祖 最初は，統合開発環境といっていましたが，いまは意図的にプロフェッショナルランゲージのほうを「統合開発環境」，Quickシリーズのほうは「統合プログラミング環境」と呼んでいます．「開発」というと，初心者の方には「難しい」用語に聞こえますので，プログラミングといいかえたわけです．Microsoft CとQuickシリーズの環境とでは，出発点もアーキテクチャも違います．統合化されたプログラミング環境でも，別ものだということをアピールしたいと思っています．

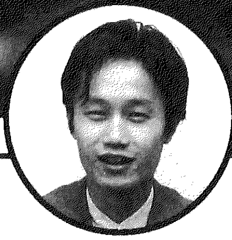
Microsoft CとQuickCの違いとは

——Microsoft CとQuickCの位置づけの違いとは？

能祖 Quickシリーズはオールインワンです．コンパイラ，デバッガ，エディタの機能が入っています．メイクユーティリティを呼び出す機能（ファイル選択するだけでメイクファイルを作成できる），アドバイザ呼び出し機能なども，基本的にはQC.EXEという実行ファイルに入っています．QuickBASICも同じです．

Microsoft C 6.0は，「プロジェクト指向」という考え方で，ツールとして個々に完全なものをそろえています．たとえば，デバッグはCodeViewを使う，コンパイラも独立したものを呼び出します．

これらのツールのベースとなって，データの受け渡しや呼び出し，あるいはメモリのスワップやディスクスワップなどをやるのがPWBです．標準で常駐するメモリ



サイズがわずか数10KBで，COMファイル1個だけです．起動すると，PWBEDというエディタを読み込んでエディットが可能な状態にします．さらに，コンパイラを，メイクを，CodeViewを呼び出します．

——プロの開発環境として発展性をもたせているわけですね．

能祖　サードパーティーのツールを，組み込むことが可能になっています．アメリカでは，すでに何本かそういった製品がでています．Windowsのように，アプリケーションインターフェースが提供されていて，Cで関数を組んで実行ファイルを作ります．PWBは，ある特殊な拡張子がついていると(MXTもしくはPXT)，それを自分が呼び出せるファイルだと認識します．

基本的にはマイクロソフトのツールで満たされるようにしたいと思いますが，それだけではどうしても全部を提供しきれないところがあります．たとえば，MIFESやVz, Final, REDなどを使い慣れている方が大勢います．PWBが提供している機能を使えば，PWBのエディタと完全に入れ替えることもできるようになるでしょう．

PWBはOS/2上で快適に動く

——プログラマの方からはOS/2で使うPWBは快適だと聞きますが？

能祖　アメリカでは，プログラマが使うOSとしてOS/2が非常に有効だとされています．日本とは状況が違って，ハードウェアがPC/ATだけであり，MS OS/2とPWBのフェーズがとりやすかったということがあります．日本でも，OSの開発はしていますが，アダプテーションの細かいところ，出荷，サポートなどは各OEMのメーカーに行っていただくので，どうしてもMicrosoft C 6.0という商品とフェーズが合わないところがでてきます．たとえば，DOS互換ボックスでアプリケーションを開発して，DOSの3.0以上で本当に互換性をもって動かせるかというと，必ずしもそうではありません．「OS/2はDOSのアプリケーションを作るために使えます」といっているのですが，実際に使っている方はまだ少ないです．

PWBは，DOSではどうしても重たいので(将来的には解決できると思いますが)，いまのところは286，386以上のCPUでプロテクトメモリを利用できる環境でないと使いやすいとはいえません．軽さという点では，Quickシリーズのほうがよほど使いやすいでしょう．

——PWBは，OS/2に照準を合わせて作られたと考えてよいのでしょうか．

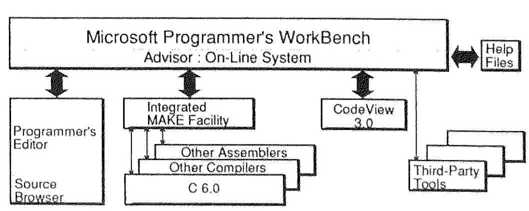
能祖　一部の機能でそれがとくに重要視されているところはありますね．PWB上でコンパイラを呼び出すときに，OS/2はバックグラウンドでできます．しかも，エディットの途中でもコンパイルが終了するとビープ音が鳴って，COMPOSEを開く．ところがDOSは，コンパイルが終わるまで待たなくてはいけない．大規模なプログラムの開発ができるなど，DOS上でもメリットはあるんですが，やはりOS/2上で高く評価されるところを見ると，いまのPWBはOS/2で使うのがベストだと思っています．

90年代はオブジェクト指向型ツールへ

——PWBの今後の展開はどうなるのでしょう？

能祖　いまのところ，PWBが対応しているのはCだけです．ほかの言語製品へのPWB対応を今期のうちにやっておきたいと心がけています．

PWB自体を新しくするということも考えています．まず，DOS上では遅いというご指摘を改善していきたいと考えています．それと「軽さ」ですね．PWBからコンパイラを呼び出すとき，PWBからCodeViewを呼び出すときも，スピーディな操作を実現できるようにしたい．



そして，大規模なプロダクツを開発できるようにしたい．たとえば，PWB自体をハイメモリに追い出して，メモリをユーザーのプログラムに明け渡すとか……．

もっと先のことをいいますと，昨年発表したAdvanced Development Environmentがあります．90年代の流れとして，オブジェクト指向型の開発ツールを考えています．統合したかたちで，各ツール間はプログラム間の情報（メッセージ）を伝えて，個々のツールのインターフェースを小きざみにしていこうと考えています．いまは，PWB側でソースを作ってコンパイルしますね，できた実行ファイルをCodeViewでデバッグする．デバッグ情報(ブレイクポイント，トレースを行った位置など)を残したままPWBにもどります．それでソース行を変更しても，CodeView側でそれをちゃんと認識できるようにしています．これをもっとフレキシブルに細かくできるように，QuickCのなかでやっているのと同じくらいのイメージでできるようにしたいということです．

また，各ツールのパフォーマンスをあげることも考えています．PWBの次のバージョンでは，ツールの機能を拡張したものを提供できると思います．その先では，各ツール間のインターフェースをもっと細かく，情報をすべてデータベースのようなものに蓄えるとか，オブジェクト指向型のデータベースにしたりとか，そういうところまで考えています．

ボーランドの製品へ着々と迎撃の準備

——Windowsアプリケーション開発用の製品はどうでしょうか？

能祖　いまのところ，Windowsアプリケーションの開発に対応した製品はMicrosoft Cしかありません．アメリカでは，Borland，Zortechなどが加わって複数になっています．Borland C++だけはSDK（Software Development Kit)が不要です．米国マイクロソフトでは，SDKバンドルパッケージを企画しています．日本では制約があって，今回は見送るつもりですが，近い将来にはSDKなしでWindowsアプリケーションが開発できる環境を提供したいと思っています．

——言語製品に関する具体的な予定は？

能祖　Turbo Pascal 6.0，Turbo C++などのオブジェクト指向のプログラミング言語を先行されましたから．その迎撃を次のMicrosoft Cでやりたいと思っています．ボーランドのC++を綿密に見たうえで開発しますので，現状でボーランドのC++に満足されない方でも十分満足していただける環境になると思います．

Windows上の開発環境は，Turbo Pascalで実現され，Cでもやるでしょう．これも早いうちに迎撃したいと考えています．先行を許してしまいましたが，開発力・企画力の差ではありません．

C++は，市場に出すのはまだ時期が早かったと思うんです．AT&Tで2.0がでて，すでに2.1がでようとしている．私どもも，2.0ベースでやるつもりなんですが，2.1ではある程度のスペックが見えていますので，マイクロソフトの拡張機能として，そのうちのいくつかをご提供できると思います．ユーザーからみれば，いったんC++に移っても，バージョンが変わるたびにクラスライブラリなど，過去の資産を変更しなければならないことになれば，C++のオブジェクト指向に移す意味がなくなりますから．

たしかにTurbo C++は，OOPの勉強には適していると思います．全世界のCのユーザーは手続き型からオブジェクト指向型へまだ移行されていないでしょう．なぜなら，オブジェクト指向プログラミングはOOP（Object Oriented Program）だけではなく，OOA（Object Oriented Analysis）や，OOD（Object Oriented Design）も含まれるからです．雑誌などでも，どんどんそういった概念をだしていただければと思います．

プロフェッショナル向けに新たな機能強化

Turbo C++/Borland C++のIDE

株式会社ボーランドジャパン 大野元久

昨年暮れに日本でもTurbo C++が発売され，いよいよ本格的なオブジェクト指向プログラミング時代の幕開けかと騒がれたのはまだ記憶に新しいところだが，今年3月，アメリカではいち早くソフトウェア開発キットなしで，Windows3.0用アプリケーションが開発できるBorland C++が発売された．Turbo Pascal以来，統合開発環境には強いボーランド社であるが，新しい2つの製品で，統合開発環境IDE（Integrated Development Environment）はどのように改良されたのだろうか．ここでは，画面やメニュー，操作方法などのユーザーインターフェースを中心に，新しいIDEの環境を解説し，IDEのトランスファ機能を利用するサンプルプログラムを紹介する．

一般に，プログラムを開発する作業は「プログラムの編集」「コンパイル，リンク」「デバッグ」の3段階にわけられます．アプリケーション開発においてもっとも望ましい状態とは，プログラマが「プログラムの編集」や「デバッグ」に集中できる環境が提供されることです．コンパイルやリンク，あるいは3つの段階の切り換えによって肝心の作業そのものが分断されるようでは，せっかく浮かんだアイデアを実現する妨げになったり，思考を中断させてしまうことにもなりかねません．

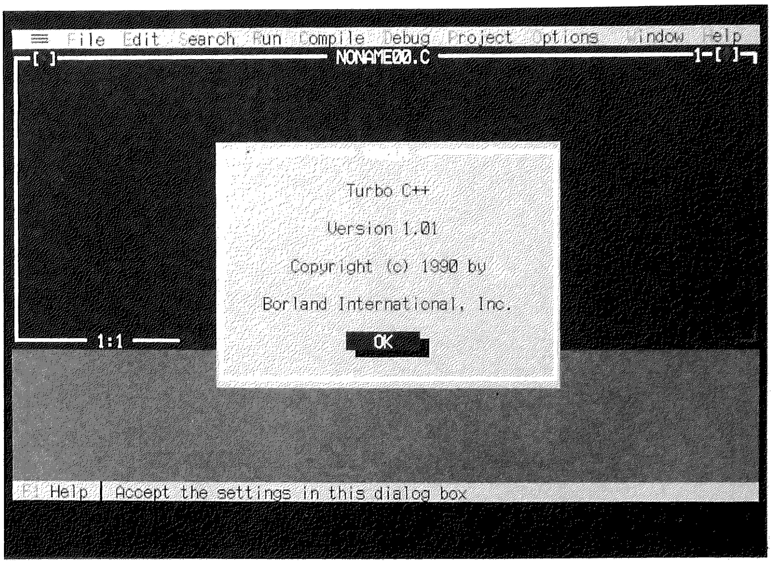
Turbo C++/Borland C++のIDE (Integrated Development Environment)では，これら3段階をすべてIDE自体がサポートしているため，それぞれの作業を切り換えるための待ち時間はありません．そのうえ個々のツールにも改良がほどこされています．IDEは，もはや初心者用の簡易ツールではありません．この記事によって，プロフェッショナル・プログラマの方々の興味をひくことができれば幸いです．

はじめに

プロフェッショナルを意識した機能改良

これまでIDEは，しばしばプロフェッショナル・プログラマに批判されることがありました．そのもっとも大きな理由は単独のツールに比べて機能が不足しているという点です．くわえて，既存のツールとの操作性の違いが受け入れられていなかったという点もあげられるでしょう．たしかに，Turbo Cをはじめとするこれまでの製品に付属するIDEにはそうした側面があったといえます．大きさの制限されたファイルを1つしか編集できない，デバッグの機能やメモリサイズが制限されている，といったことは，複数の分割ファイルで大規模なアプリケーションを開発する人にとってはIDEから得られるメリット以上に大きなデメリットでした．

Turbo C++/Borland C++に付属する新しいIDEは，プロフェッショナルプログラマの使用に耐えうるよう，すべての機能が大幅に改善されています．IDEには，市販のエディタにひけをとらない機能を備えた内蔵エディタ，CとC++の両方をサポートしたコンパイラ，オーバーレイをサポートしたリンカ，改良されたデバッガが組み込まれています．また，新しいIDEは各種のカスタマイズ機能とトランスファによる外部プログラムの組み込みによって，既存のツールを利用しながらIDEを使うことが可能です．それでは次に各ツールと，新しいIDEの特徴を紹介します．



画面1 IDEの起動画面

エディタ

専用エディタをしのぐ機能も

新しいIDEは従来のTurboシリーズのIDEを全面的に書き換えたものです．なかでもエディタを使用する時間は，プログラマがアプリケーション開発に費やす全工程のうちでかなりの部分を占めます．新しいエディタの機能と操作性によって，プログラマがIDEを利用する機会は大幅に広がるでしょう．

◆マウスのサポート

新しいIDEではマウス操作がサポートされています．基本的な操作はマウスの左ボタンに割り当てられており，メニューやダイアログボックスの選択操作，ブロック指定，ウィンドウの位置や大きさの変更などさまざまな編集操作を処理できます．右ボタンは，へ

統合開発環境用語集
デバッガ プログラムを実行しながら，エラーの原因を探す手助けをするプログラム

ルプの呼び出しやブレークポイントの設定など特殊な機能を割り当てておきます.

98版では，マウスドライバはIDEに内蔵されているため，組み込まれているマウスドライバがMS形式かNEC形式であるかを気にする必要はありません.

◆編集できるファイル数とサイズ

従来，TurboシリーズのIDEのエディタは，編集できるファイルが64KBに制限され，同時には1つのファイルしか編集できませんでした．複数のファイルを編集するためにピックファイルという機能で，以前に編集したファイルを簡単に呼び出すことができた程度でした．新しいIDEでは，合計8MBまでの複数のファイルを編集することができます．編集できるファイルの数はメモリ容量にのみ依存し，事実上制限はありません．編集できるプログラムの大きさに関しては，問題はなくなったといってよいでしょう.

新しいIDEでは，オーバーラップ可能なマルチウィンドウがサポートされているので，複数のファイルを編集する場合も，画面上の好きな位置に好きな大きさでウィンドウを配置できます．もちろんタイル形式に配置することもできます

複数のファイルを編集できるようになったことで，分割されたプログラムの関連する箇所を参照したり，他のテキストファイルを参照しながらプログラミングするという，いわば市販のエディタではあたりまえのことができるようになったわけです.

◆異なるウィンドウでの同一ファイルの編集

同一ファイルを異なるウィンドウで編集できる点も重要です．これは，シンボルの宣言部分を1つのウィンドウで参照しながら，同じプログラムでそれを参照している部分を編集するような場合に使えます．プログラミングにおいてはかなり重要な機能だと思われます.

◆キャッシュ機能

エディタにはキャッシュ機能が組み込まれていて，ファイルのどの部分も瞬時に移動できます．この機能は日本で市販されているほとんどのエディタには備わっていませんが，非常に有益な機能のひとつです．たとえば，タグジャンプと呼ばれるエラーが発生した行へ移動する機能では，ファイルの先頭でも最後でも瞬時に移動できるため，「エラーの発生→プログラムの修正」という作業をスムーズに行えます.

◆検索時の正規表現，全角半角の同一視

文字列を検索する際に，GREP形式の正規表現を利用できます．たとえば，'['と']'で囲まれた文字列 ([. *]) とか，数値を引数としているprintf (printf. * [0-9]+) を検索することができます．全角半角を同一視する機能は日本語版で追加された機能で，英数字の全角と半角を区別しないで検索するためのものです．英大小文字を同一視する機能と併用すれば，STRING, string, Stringのすべての文字列を検索の対象とすることができます．全角や半角を混在させて記述されているコメントを検索する場合などに便利でしょう.

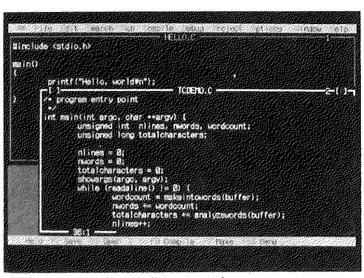
◆入力ヒストリー機能

オープンするファイル名の入力や文字列検索など，文字列を入力するか所のほとんどでヒストリー機能を使うことができます．ヒストリー機能とは，以前に入力した文字列を再入力する手間をはぶき，リスト表示されている文字列の中から選択できるようにする機能です.

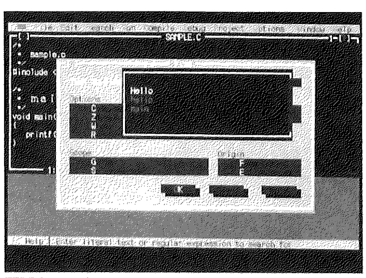
プログラムを作成するときには，同じような処理を何度も繰り返さなければならないことがしばしばあります．IDEのエディタは，さまざまな場面でできるだけ少ない手間で開発ができるように設計されています.

◆キーカスタマイズ

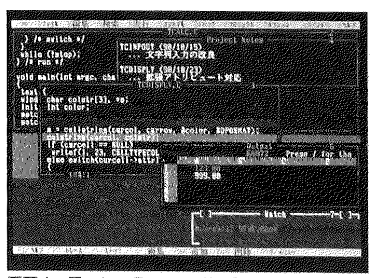
慣れ親しんだエディタからIDEのエディタに移行する際にキー操作がまったく異なっていては，操作を最初からおぼえなおさなければなりません．多



画面2　マルチファイルエディタ



画面3　入力ヒストリー



画面4　Turbo C++ハイレゾリューション

統合開発環境用語集
ドラッグ　ボタンを押しながらマウスを移動させる操作．ブロックの始点から終点まで指定する場合などに使う.

くのエディタがそうであるように，IDEのエディタもキー割り当てを好きなように変更することができます．TEML（Turbo Editor Macro Language）を使えば，複数の処理や定型テキストを1つのキーに割り当てておくこともできます．

◆完全なプロファイル機能

一般にプロファイル機能とは，エディタを終了するときに編集中のファイル名や行番号を保存しておき，次回エディタを使用するときにこれらを復元する機能をいいます．Turbo C++/Borland C++のプロファイル機能は，こうした単純なものではありません．ファイル名や行番号はもちろん，ウィンドウの配置，検索文字列，カットしたブロックの内容，エディタオプションなど各種の設定のすべてが保存，復元されます．このためIDEを終了しても，次に起動するときに終了前と同じ環境をそのまま復元できるわけです．Borland C++では，ウオッチ式やブレークポイントも保存できるようになっており，どの内容を保存するか選択することができます．

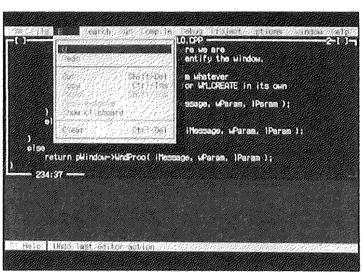
◆ハイレゾリューションモード H98の拡張アトリビュート対応

ハイレゾリューションモードやH98の拡張アトリビュートに対応しているため，幅広い環境で本体のもつ機能を十分に活用することができます．

Borland C++では，Turbo C++にさらに次の機能が追加されています．

◆アンドゥ，リドゥ

Borland C++のIDEでは，すべての編集操作のアンドゥ（取消），リドゥ（再実行）が可能になっています．これは，単に削除した文字列やブロックを復元するとか，ファイルを編集前の状態に戻すというレベルのものではありません．IDEは，メモリの許す限り文字やブロックの削除以外に，カーソルの移動，上書きした文字列，置換などあらゆる編集操作をすべて記録しています．そして，すべての編集操作を逆順に取り消すことができるのです．置換文字列をまちがえてしまった場合でも，アンドゥ機能によって正しく以前の状態を復元できるわけです．グループアンドゥによって同じような処理を一度に復元することもできます．取り消しすぎた場合は，リドゥ機能で必要な位置まで再実行させることができます．



画面5　Borland C++の画面

◆出力画面の取り込み

Turbo C++では，ユーザープログラムの出力画面は表示することしかできませんでしたが，Borland C++ではこの内容をテキストとして取り込むことができるようになっています．

コンパイラ，リンカ

仕様と性能はコマンドライン版と同じ

IDEのコンパイラ，リンカは，コマンドライン版と同じ仕様のものです．以前は，IDEではインラインアセンブラが使えないという制限がありましたが，Turbo C++ではこの制限はなくなりました．さらに，Borland C++では組み込みアセンブラ機能が追加されて，コマンドライン版のアセンブラがなくてもインラインアセンブラが使えるようになっています．

オプションの設定，変更もマウスを

メイクからコンパイル開始までの時間を短縮する

Turbo C++は，Turbo C 2.0に比べてプロジェクトメイクでコンパイルを開始するまでの時間がかかっているのではないかという話を聞きますが，これはオブジェクトの自動依存情報を検査しているためです．Turbo C++では，ソースファイルがインクルードしているヘッダファイルの変更に関する情報をオブジェクトに埋め込まれた自動依存情報をもとに検査します．このため，すべてのオブジェクトを調べるために若干時間がかかっているわけです．

ヘッダファイルはほとんど変更することがないという場合は，自動依存情報のチェック（Options | Make | Check auto-dependencies）をオフにしてください．これで，メイクを選択してからコンパイルを開始するまでの時間が短縮されます．ただし，安易にこのオプションをオフにしてしまうとヘッダの修正がコンパイルに反映されず，見つけにくいバグを発生させてしまうこともありますので注意してください．

統合開発環境用語集
日本語処理　全角文字は2バイトで表現されるため，日本語文字操作では特別な操作や判断が必要な場合がある．

使ってメニューから選択できますし，それぞれのオプションについてヘルプ画面が用意されていますので，いちいちマニュアルを開く必要はほとんどなくなるでしょう．

コンパイルの性能は，通常はコマンドライン版と同じです．EMSメモリやプロテクトメモリがあれば，IDEはオーバーレイ領域としてこれらの拡張メモリを使用することができます．再コンパイルするときは，オーバーレイコードがこれらの拡張メモリにロードされているためコンパイル時間が若干短縮されます．さらに，Borland C++ではTurbo Driveを利用してプロテクトメモリにIDEをロードして実行するバージョンが用意されています（コマンドライン版もあります）．このバージョンでは，オーバーレイによるオーバーヘッドがないため，高速にコンパイルできるばかりか十分な大きさのプロテクトメモリがあればかなり大きなソースプログラムをコンパイルすることもできます．

コンパイルにIDEを使う最大のメリットは「コンパイルエラー→ソースコードの修正」という作業を一切中断されずに行うことができる点です．市販されている多くのエディタでも，エラーメッセージを使ったタグジャンプ機能をサポートしていますから単純なエラー行番号の追跡を行うことはできるでしょう．しかし，修正を行ったために行番号が変更されてしまうとエラー行の追跡が難しくなります．IDEを使えば，ソース行を変更しても他のエラーの発生した位置を確実に追跡できます．また，エラーの内容についても即座にヘルプ画面を呼び出して確認することができます．



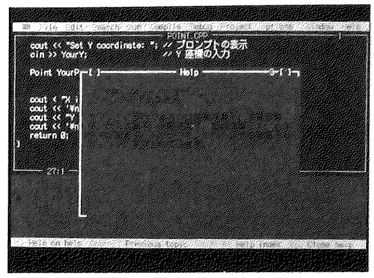
画面6　エラーの追跡

デバッガ

より広いメモリ空間を提供

IDEからはTurbo Debuggerを呼び出すこともできます．しかし，「だからIDEにデバッグ機能はいらない」といえるでしょうか．はじめに述べたように，作業の切り換えに手間取っていては開発効率によい影響を与えません．もしIDEからデバッガを呼び出してすませるならば，この点に関しては通常のエディタで編集し，コマンドラインでコンパイルして実行したりデバッガを使ったりするのと大差ありません．

IDEのデバッガは，Turbo Debuggerのようにさまざまな手法で省メモリデバッグをサポートしているわけではありませんが，ボーランド独自のオーバーレイ機能VROOMMの採用によって，Turbo C 2.0よりも広いメモリ空間をユーザープログラムで利用できます．デバッグのための機能も，単なるブレークポイントやウォッチ式の設定ばかりでなく，ブレークポイントでの条件式，パスカウントが設定でき，複雑なデータを追跡しやすくするためのインスペクト機能，式の評価／変更，呼び出しスタックの追跡など，ソースレベルでのデバッガとしては十分な機能がサポートされています．式の評価



画面7　ヘルプ画面

や条件式の設定では，C式をそのまま指定することができます．アセンブルレベルでのデバッグは，レジスタウィンドウがサポートされているだけですが，ある程度の規模のアプリケーションやテストモジュールのデバッグには十分実用的でしょう．

IDEの中にいるわけですから，実行中にプログラムミスを見つけたらその場で修正できます．作業の切り換えに余計な時間は必要ありません．より詳細なデバッグを行いたい場合は，後述のトランスファ機能を使ってTurbo Debuggerを呼び出せばよいのです．

◆インスペクト機能

インスペクト機能は，Turbo Debuggerで実現されたデータデバッグを支援するための機能です．配列や構造体をインスペクト（調査）するとインスペクタウィンドウに各要素が表示されます．ここで，カーソルを適当な要素の上に移動させると，その要素についてさらに詳しい情報が表示されます．その要素が単純なデータ型（スカラ型など）であれば，その内容を変更することもできます．

インスペクト機能は，Turbo Debuggerの大きな特長のひとつだったのですが，これがTurbo C++/Borland C++にも採用されたことで，ネストされた構造体型をもつデータやリスト形式のデータを非常に簡単に追跡できるようになっています．C++

統合開発環境用語集
入力ボックス　ウィンドウの一種で，ファイル名などの文字列の入力を受け付けるのに使われる．

モードでは，クラスの親クラスやメンバ関数も表示されます．

トランスファ機能

外部プログラムをIDEに組み込む

トランスファ機能とは，IDEから他のプログラムを呼び出すための機能です．しかし，トランスファ機能の本質はDOS Shellからプログラムを呼び出す肩代りをすることではありません．この機能を使えば，外部プログラムをIDEに内蔵された機能と同じように使うことができるのです．つまり，トランスファ機能によって新しいIDEはProgrammer's Platformと呼ぶに値するものとなったのです．

たとえば，アプリケーションの中にはアセンブラモジュールをリンクしなければならないものも少なくありません．Turbo C 2.0では，このようなアプリケーションを構築しようとすると，アセンブラモジュールをコマンドラインやDOS Shellであらかじめアセンブルしておかなければなりませんでした．トランスファ機能を使えば，プロジェクトに含まれるソースコードをコンパイルするために，統合環境ではサポートされていないアセンブラやコンパイラを指定することができます．アセンブラが出力するエラーメッセージもメッセージフィルタを通してIDEに取り込むことができ，IDEでコンパイルした場合と同じようにエラーを追跡することができます．

このメッセージフィルタのソースコードは公開されているので，特殊なコンパイラなどがあっても，専用のフィルタを作ってIDEの機能の一部として組み込むことが可能です．実際，Borland C++ではリソースコンパイラとインポートライブラリアンがトランスファプログラムとして組み込まれており，Windowsアプリケーションのプロジェクトの一部を構成するリソースなどを扱うために使われています．

また，どうしてもこれまで使ってきたエディタでプログラムを編集したいという場合にもトランスファ機能を使うことができます．トランスファプログラムには，編集中のファイル名や行番号をマクロを使って指定できるため，Editウィンドウ中のイメージをそのままエディタに渡すことができます．

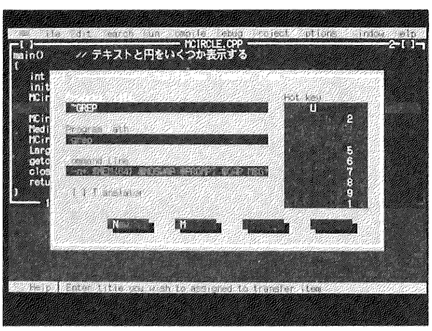
トランスファ機能でエディタを設定する例

VZエディタ（COMMAND.COMを仲介させた場合）

```
Program Title
[VZ EDITOR (resident mode)                    ]
Program Path
[COMMAND.COM                                  ]
Command Line
[SSAVE CUR /C VZ $EDNAME -$LINE               ]
```

MIFES

```
Program Title
[MIFES EDITOR                                 ]
Program Path
[MIFES.EXE                                    ]
Command Line
[SSAVE CUR $EDNAME+$LINE                      ]
```



画面8 トランスファプログラムの設定



画面9 トランスファプログラムの実行

統合開発環境用語集
ネスト　複文・反復制御・選択制御構造・条件組み込みなどに存在する．ソースプログラミングでは目で判断できるようインデントを行う．

プロジェクト機能

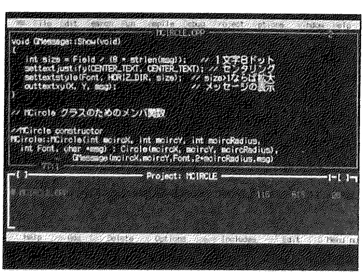
ファイル名からオプション，レイアウトまで

プロジェクト機能は，従来のIDEから飛躍的に改良された機能のひとつです．プロジェクトファイルが管理するものは，プロジェクトを構成するファイル名ばかりではありません．必要なコンパイラオプション，リンカオプション，ウィンドウレイアウトなど，アプリケーション開発におけるすべての環境をプロジェクトごとに管理できます．個々のプロジェクトはまったく独立して管理されており，他のプロジェクトが使用しているコンパイラオプションや環境には影響されません．

プロジェクトを構成するファイルについては，個別にコンパイルオプションを指定できるほか，インクルードされているヘッダファイルを参照したり，コンパイルするためのプログラムを指定することも可能です．アセンブラモジュールを組み込めば，そのモジュールは自動的にアセンブラに渡されて処理されます．オーバーレイの指定やデバッグ情報の有無なども簡単に選択できます．

IDEは，オプションやウィンドウレイアウトなどの環境をプロジェクトごとに管理しますが，新しいプロジェクトをオープンするときに限っては以前の環境を引き継ぎます．たとえば，単一のソースコードからだんだん規模が大きくなり複数のファイルに分割することになった場合も，その環境を引き継いだまま新しいプロジェクトをオープンできるわけです．設定を初期化したい場合は，システムメニューでClear desktopを選択してください．

トランスファ機能と組み合わせて，オブジェクトファイルの作成を行わないものでも自動化作業の中に組み込むことができます．たとえば，Borland C++でWindowsアプリケーションを作成する場合，プロジェクト機能は非常に役立ちます．アセンブラモジュールに対してアセンブラが呼び出されるのはTurbo C++と同じですが，リソースコンパイラを呼び出して.RCファイルを.RESファイルにコンパイルしたり，.RESファイルを.EXEファイルにバインドすることもできます．インポートライブラリアンを呼び出すための機能も用意されています．



画面10 Projectウィンドウ

ウィンドウ

各種情報はウィンドウ単位で表示

IDEには，アプリケーション開発を支援するためのさまざまなウィンドウが用意されています．Turbo C++/Borland C++のIDEでは，オーバーラップマルチウィンドウがサポートされており，ウィンドウの数やその位置と大きさは自由に変更できます．マウスがサポートされているためこれらの変更は容易に行うことができます．また，ウィンドウのレイアウトはプロジェクトごとに管理されます．

◆Editウィンドウ

ソースプログラムを編集するためのウィンドウです．プログラム以外にもドキュメントファイルを編集することができます．機能の詳細については，「エディタ」のところを参照してください．

◆Messageウィンドウ

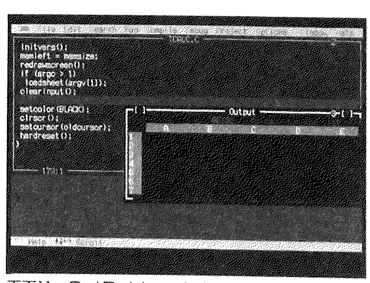
コンパイル，リンク時に発生したエラーや警告メッセージを表示します．トランスファプログラムの出力をメッセージフィルタを使って，このウィンドウのために変換して利用することもできます．

◆Outputウィンドウ

プログラムの出力を確認するためのウィンドウです．IDEの画面とユーザー画面をまったく切り換えてしまうのでなく，IDEのウィンドウ中にユーザー画面を表示できるためデバッグの際にとくに効果的です．

◆Watchウィンドウ

ウォッチ式を指定したり，指定した式の値を表示するためのウィンドウです．構造体や配列を指定すると各要素を1行に並べて表示します．



画面11 OutPutウィンドウ

統合開発環境用語集
引数 関数や手続きの呼び出し時，引数で値を渡す．

◆Inspectorウィンドウ

詳しく調査したい変数や型などを表示するためのウィンドウです．構造体や配列を指定すると各要素を行ごとに分けて表示します．適当な要素を選択すれば，さらに詳しい内容を調べることができます．

◆Registerウィンドウ

プログラム実行中のレジスタの値を表示します．IDEのデバッガにはアセンブルレベルでのデバッグをサポートする機能はありませんが，インラインアセンブラをCのソースレベルでデバッグすることはできます．また，特定のレジスタをウォッチウィンドウに表示しておきたい場合は，レジスタ疑似変数（_AX，_DLなど）を使うことができます．

◆Projectウィンドウ

プロジェクトを構成するファイルを指定したり，各構成ファイルに対するオプションを指定するためのウィンドウです．

◆Project notesウィンドウ

開発中のプロジェクトに関する開発履歴やコメントなどのドキュメントを記録しておくウィンドウです．

省メモリ化
独自オーバーレイ機能 VROOMM

新しいIDEには，VROOMMというボーランド独自のオーバーレイ機能が使用されています．このため，数多くの機能を追加したにもかかわらずTurbo C++が消費するメモリはTurbo C2.0よりも少ないくらいです．

さらに，Borland C++ではTurbo DriveというDOS-Extender技術を使って統合開発環境を動作させることができます．実装されているプロテクトメモリをすべて有効に使うことができ，大規模なアプリケーション開発にも適しています．また，ここからTD286を呼び出すこともできるため，デバッグ環境においてもユーザープログラムが広いメモリ空間を利用できます．

IDEのちょっといいテクニック
メモリ節約の工夫

新IDEがいくら省メモリだといっても，やはりIDEを終了してからプログラムを実行するほうがよほどメモリに余裕があります．Turbo C++のIDEには，現在の環境のほとんどすべてをディスク上のファイルに記録しておく機能がありますので，これを利用してIDEを終了させて目的プログラムを実行し，再度IDEに戻ってくる環境を作ってみましょう（PC-9801専用）．

あらかじめPPF_SUB.Cをスモールモデルでコンパイルしておき，PPF.BATと一緒にTC.EXEと同じディレクトリにコピーします．コマンドラインから，

```
A>PPF
```

と入力すると，Turbo C++のIDEが起動しますので，Options | Transferを選択してトランスファ項目を図1のように設定します．

Hot Keyを適当に割り当てておくと便利です．ここでは，[Shift]+[F6]を割り当てたものとします．さらに，Turbo C++のOptions | Environment | PreferencesメニューのAuto Saveオプションのうち，Environment, Desktop, Projectがチェックされているかどうかも確認してください．これがチェックされていないと，再度Turbo C++のIDEに戻ってきたときに以前の環境が復元されません（なお，カレントディレクトリに複数のプ

```
Program Title
[Quit|Turbo Debugger                          ]

Program Path
[PPF_SUB.EXE                                  ]

Command Line
[$SAVEALL $NOSWAP $MEM(64) -Q TD $EXENAME]
```

図1

統合開発環境用語集
ヒストリ　履歴ともいう．キー入力や画面の変化などを過去にさかのぼって再現する機能

リスト1 PPC.BAT

```
ECHO OFF
IF EXIST PPF_CMD.BAT DEL PPF_CMD.BAT
TC %1 %2 %3 %4 %5 %6 %7 %8 %9
IF EXIST PPF_CMD.BAT PPF_CMD

```

リスト2 PPF_SUB.C

```
/*
 *  Programmer's Platform
 *  written in Turbo C++(small model)
 */
#include <stdio.h>
#include <string.h>
#include <dos.h>

unsigned _stklen = 1000;
unsigned _heaplen = 1000;

main(int argc, char **argv)
{
  static char ppf_cmd[] = "PPF_CMD.BAT";
  int quit;
  FILE *fp;

  if (argc > 1 && stricmp(argv[1], "-Q") == 0) {
    --argc;
    ++argv;
    quit = 1;
  } else {
    quit = 0;
  }

  fp = fopen(ppf_cmd, "w");
  if (fp == NULL) abort();
  while (--argc) {
    fputs(*++argv, fp);
    fputc(' ', fp);
  }
  fprintf(fp, "¥nPPF¥n");
  fclose(fp);
  disable();
  if (quit) {
    *(unsigned short far *)MK_FP(0, 0x502) = 0x6B00;  /* F10 */
    *(unsigned short far *)MK_FP(0, 0x504) = 0x6B00;  /* F10 */
    *(unsigned short far *)MK_FP(0, 0x506) = 0x2046;  /* 'F' */
    *(unsigned short far *)MK_FP(0, 0x508) = 0x1051;  /* 'Q' */
    *(unsigned short far *)MK_FP(0, 0x524) = 0x502;   /* kbdbuf */
    *(unsigned short far *)MK_FP(0, 0x526) = 0x50A;
    *(unsigned char far *)MK_FP(0, 0x528) = 4;
  }
  enable();
  return 0;
}

```

ロジェクトファイルがある場合も以前のプロジェクトの状態に復帰できません).

さあ，これで準備はできあがりです.

エディタのプログラムをコンパイルしたら，[Shift]+[F6]を押してください．しばらくトランスファのためのウィンドウが表示された後，Turbo C++が自動的に終了しTurbo Debuggerが呼び出されます．このとき，デバッグ情報をオンにしてコンパイルすることを忘れないでください．また，プログラムがコンパイルされていないとTurbo Debuggerがファイルを見つけられません．アクティブウィンドウにドキュメントファイルが表示されていたりすると，正しく動作しません．こうした場合の対処については，皆さん自身でPPF_SUB.Cを書き換えてみてください.

ここで呼び出されたTurbo Debuggerは，単にバッチファイルから呼び出されているわけですからIDEによるメモリの負担がありません．トランスファメニューのCommand LineでTDの代わりにTD286とかTD386としておけば，それらを呼び出すこともできます.

PPF.BATは，Turbo C++のIDEを起動しますが，IDEが終了した時点でPPF_CMD.BATというバッチファイルがあれば，そのバッチファイルを実行します．PPF.BATは，IDEを呼び出す前にPPF_CMD.BATが存在していると，このファイルを消去します．PPF_CMD.BATは，IDEにいる間に作成されるわけです.

このバッチファイルを作成するのが，PPF_SUBです．PPF_SUBは，コマンドライン引数で指定された文字列ともとのPPF.BATを呼び出すための文字列をPPF_CMD.BATに書き込みます．コマンドライン引数の先頭に-Qオプションが指定されていると，キーボードバッファにF10，F10，F，Qという文字列を書き込んでから終了します．これによって，トランスファからIDEに制御が移されたときに，自動的にメインメニューからFileメニューが選択されQuitすることになります．この部分に関しては機種やFEPによって動作しない場合も考えられますので，うまく動作しない場合はトランスファメニューのCommand Lineから-Qオプションを除き，トランスファ実行後に[GRPH]+[X]などでIDEを終了するようにしてください.

IDEは初心者ためのものというイメージがぬぐいきれない方もまだ多いと思います．しかし，いまやプロフェッショナルだからこそIDEを使う時代です．プロフェッショナルプログラマであれば，より快適で効率のよい開発環境を貪欲に望んでいるはずですし，現在のIDEはそれに応えるだけの機能と性能を備えているといえるでしょう.

INTERVIEW

ボーランドジャパンに聞く

Windowsが製品のキーワードになった

株式会社ボーランドジャパン 企画部長

篠原幸吾

Borlandの統合開発環境の発展

——Borlandといえば，統合開発環境というくらい，切っても切れないイメージがありますが，BorlandがTurbo Pascalを最初に出されたときは，教育向け，プロフェッショナル向けという特別な意図があったのでしょうか．

篠原　ユーザーを限定するような，そうした考えはなかったでしょう．プロフェッショナル用として通用するものを，教科書なみの価格で発売するということで，すべてのユーザーが対象となっていたといえます．その考え方が，多くのユーザーに理解されて成功につながったことは十分に証明されているでしょう．

現在では，64KBのメモリにすべてがおさまっていた時代とは違って，プロフェッショナル向けのものがどんどん高機能かつ専門化してきています．Borlandも，Borland C++とTurbo C++のように，プロフェッショナル向けのものとビギナー向けのものを明確にわけるようになりました．しかし，初心者だから機能の低いものを提供するということではありません．何を学ぶにしても，つねに本物のよい道具を使って学ぶのが最適だからです．Turbo C++は十分に実用に耐えるコンパイラですが，ビギナーの方にはとくに必要ないと思われるものを取り除いて入手しやすい価格で提供しています．

——エディタつきのコンパイラという発想は，どこから思いついたのですか．

篠原　エディタつきのコンパイラという発想はそれ以前にもあったように思います．ただTurbo Pascalが画期的だったのは，コンパイル中にエラーがあるとその位置をエディタ中で表示できたことでした．これは，当時Turbo Pascalのコンパイル速度が驚異的といわれるほど速かったからできたことです．

——現在，Borlandは統合開発環境をどのように位置づけているのでしょう．

篠原　もちろん，すべてのユーザーを対象としています(笑)．統合開発環境（Integrated Development Environment:IDE）という言葉はTurbo Cではじめて使われたのですが，それまで統合開発環境というと内蔵されている機能がなんとなく中途半端なイメージがあったと思います．最初にTurbo Pascalがリリースされたときもそうでしたが，現在では組み込まれているものはすべて最新の機能を備えたものばかりです．

——統合開発環境は，プロフェッショナルに敬遠されがちだともいわれますが．

篠原　これはいつもいわれることですが，結局はプログラマが開発に利用し慣れ親しんだ環境と，私どもが提供する統合開発環境との違いが受け入れられないということだと思います．たしかに，Turbo　C 2.0やTurbo Pascal 5.5までは，編集できるファイルが限定されていたりコンパイルできる容量が制限されていましたが，Turbo C++やBorland C++，Turbo Pascal 6.0では全面的な改良が施されています．内蔵されているエディタ，コンパイラ，デバッガは，プログラム開発における最高水準のもので，これらのすべてを待ち時間なしで切り換えて利用できます．必要ならば外部のツールを組み込むこともできますから，これまでに使っていた開発ツールを利用することもできるでしょう．

むずかしいのは，便利なものはプロフェッショナル向けでないというようなイメージを払拭できるかどうかという点ですかね（笑）．



Windows, OS/2への対応

——Borland C++は，ようやくWindowsに対応しました．ユーザーはずいぶんと待たされていたのではないでしょうか．

篠原　バージョン2.0までのWindowsは，必ずしも多くのユーザーに幅広く使われていたとはいえません．Windows 3.0では，これまでの問題点がかなり改善され急速に普及していき，Borlandはすぐさま Windowsに対応しました．Borland C++では，オブジェクト指向プログラミングをサポートしていることで，現在では最適な開発環境を提供しているといえるでしょう．Microsoftが，C++のサポートを宣言しながら長い間もたついているのとは大違いです．日本でも，Windowsがリリースされた数か月後には，Borland C++を出荷できると思います．

——開発は順調に進んでいるわけですね．

篠原　そうですね．Turbo C++のときはTurbo C 2.0とはまったく違った環境でしたし，将来のための仕様の見直しなどを行ったためかなり時間がかかってしまいましたが，そのおかげで今回は思っていたよりも早くリリースできそうです．期待してください（笑）．

——OS/2への対応はどうなるのでしょうか．

篠原　残念ながらOS/2は，まだ幅広く使われているとはいえない状況です．Borlandは，Presentation Manager用のアプリケーション（SideKick for PM）を最初にリリースした会社ですが，Windows 3.0のように安定したものがリリースされるまでは市場を静観することになるでしょう．将来，OS/2が急速に普及するようになれば，Borlandはすぐに対応したバージョンを提供できるでしょう．

——MS-DOSの場合は640KBというメモリの制約がありますが，これはどう解決されるのでしょう．

篠原　Borland C++では，Turbo Driveという DOS-Extender技術が統合開発環境やコマンドラインコンパイラに使われています．これは，80286や386のプロテクトモードを使って広いメモリ空間を利用するものです．メモリさえあれば，統合開発環境から Turbo Debuggerを呼び出しても，消費されるメモリがわずかですみます．

——Turbo Driveの技術をユーザープログラムから利用できるようにはしないのですか．

篠原　現在はまだ考えておりません．いまは社内で利用しているわけですが，まだまだどんどん進化し変化しているテクノロジーですから．ユーザーに公開するにはまだ早いと思います．

Turbo Pascalとオブジェクト指向

——最近，Turbo Pascal 6.0の日本語版が発売されましたが，Turbo Pascalは，どのようにWindowsに対応していくのでしょうか．

篠原　詳細はまだお話できませんが，近々，Turbo Pascal for Windowsという製品が発売されます．これは，統合環境がWindows上で動作し，Windowsアプリケーションを開発するためのものです．Windows用のクラスライブラリも付属しています．SDKは不要です．

——Turbo Pascal 6.0に付属しているTurbo Visionとはどのようなものでしょうか．

篠原　Turbo Visionは，テキストベースでのウィンドウシステムをサポートするクラスライブラリです．これまでにも，Turboのような統合環境を作るにはどうしたらよいのかという質問はよくあったのですが，オブジェクト指向を利用して使いやすく構築されたウインドウライブラリはありませんでした．Turbo Visionは，まさにTurboPascal 6.0で使われているウィンドウシステムですから，実用性は十分に証明されています．

オブジェクト指向がよくわからないという方は，Turbo Visionを使って学習することができるでしょう．アプリケーション開発者にとっても，使いやすいユーザーインターフェースをもつウィンドウシステムを構築するために苦労する必要はなくなりました．

Windowsが普及しはじめたといっても，日本には漢字というやっかいな文化がありますし（笑），まだまだア

メリカほど活用されているわけではありません．こうしたことを考えると，テキストベースのTurbo Visionはむしろ日本で受けるものかもしれませんね．

——Turbo VisionをC++に移植する予定はないのですか．

篠原　さぁ，どうでしょう（笑）．

——OBJECT VISIONとは，どのようなものなのですか．

篠原　ボーランド製品としては，最初のWindows用アプリケーションです．アメリカにはフォーム作成ツールという，日本では一般的ではないアプリケーションのジャンルがありますが，この製品はフォーム作成ツールの進化形です．つまり，Windows上で入力画面を好きなようにデザインして，それをプリントアウトしてビジネスフォームとして使ってもいいですし，この画面からデータを入力することもできるわけです．入力されたデータは当社のPARADOX形式のデータファイルやdBASE形式，ネットウェアのBTREIVE形式，テキストファイル形式に落すことができます．また，DDEリンクでほかのWindowsアプリケーションにデータを入力するためにもOBJECT VISIONは使えます．さらに，この入力画面のそれぞれの項目で用意された関数を利用して計算を行ったり，ある項目に入力されたデータに応じて次のアクションを定義するといった，プログラムを画面上でビジュアルに行う機能も備えています．米国でよく行っているデモでは，生命保険の申込システムを例にしています．生命保険の申し込み時には，いろいろな質問項目がありますし，質問の答えによって保険料が変わったりするでしょう．OBJECT VISIONを使うとこういったアプリケーションを簡単に作れます．また，同時にランタイム版も販売しますので，できあがったアプリケーションを社内にたくさん配布することも，非常にコストパフォーマンスよく行えるようになります．

Paradoxとのインターフェース

——Borlandは，最近ではビジネスアプリケーションの分野にもかなり進出しているようですが，言語製品からアクセスするようなことは考えられていますか．

篠原　Borlandが発売しているParadoxやQuattro PROは，アメリカではかなり評価が高く急速にシェアをのばしています．日本でも，先日PC-9801版のParadoxを発表しました．Borlandは，Paradoxのデータベースを中心としてQuattro PROやSideKick 2.0をはじめ，言語製品からも共通にアクセスできるよう，Paradox Engineというデータベースライブラリをサポートします．ParadoxのPAL（Paradox Application Language）では記述できないような，細かい部分や普段使い慣れた言語でデータベースを構築したい方に適しています．

——Paradoxのユーザーインターフェースは，Turboシリーズとはずいぶん違いますね．

篠原　2つの製品の生い立ちが違いますから．Paradoxは1985年にANSAという会社で開発され，その後ANSAがボーランドと合併していまにいたっています．Turboシリーズとは設計思想が違うものです．ただ，ビジネスアプリケーションの世界では，画面の最上部がメニューで，その下の行がそのメニューの説明というスタイルはこれまでもっともたくさん使われてきたユーザーインターフェースですよね．Paradoxのユーザーインターフェースもそれに準拠していますから，多くのビジネスユーザーにはスムーズに受け入れられています．ただ，Turboシリーズのユーザーは少しとまどうかもしれませんね．将来はそれぞれのWindows版がでることで，ひとつの操作性へと統合されていくでしょう．

——今後の展望など．

篠原　とにかくWindowsがここしばらくのボーランドのキーワードだと申しあげておきます．われわれには言語，データベース，表計算という3つの事業部がありますが，それぞれの事業部がすべてのリソースをWindowsに集中しています．その成果はすでにいくつかみなさんの前に登場しているわけですが，まだまだどんどんでてきますよ．米国の勢いに負けないように，ボーランドジャパンもこれまで以上にアグレッシブに製品を開発し，アグレッシブなマーケティングで世間を騒がせてみたいと思います．

特集

研究「プログラミング統合環境」

快適で効率的な開発環境を考える

人にやさしいプログラミング環境

Macintoshの統合環境とプログラミング

新居雅行

Macintoshは，ウインドウシステムとプルダウンメニューを備え，マウスによるかんたん操作を実現したパソコンとして知られ，エンドユーザーから圧倒的な指示を受けている．

しかしながら，“使って天国，作って地獄”などという言葉が流布しているように，長い間Macintoshでプログラミングを行うことは至難のことであるといわれてきた．たしかに，初期のMacintoshにおいては，そのハードウェアの貧弱さとプログラミングツールの不足からこの言葉は真実といえた．だが，現在のMacintoshにこの言葉はあてはまらない．

Macintoshの優れたユーザーインターフェースにマッチした統合環境をもつコンパイラ，インターフェース部分をビジュアルに作成できるプロトタイピングツール，ぶ厚いマニュアルを不要にしたオンラインヘルプシステムなど，Macintoshのプログラミング環境について紹介する．

最近低価格化されたことで，がぜん注目を集めているMacintosh（以下，マック）だが，DOS系パソコンのユーザーから見ればかなり異質な存在であろう．DOSを使いこなしている人がマックに対面すると，まずTYPEコマンドがないことで，しっくりこないと感じてしまうようだ．ともかく，なにかやりたいことを解決する方法が，DOSとはまるっきり違うのである．

マックはもともと「プログラミングが必要のないパソコン」であった．マックが発売された当初，なんの処理系もついてこない点をマイナス点と見たてた記事があったが，プログラミングを不必要にした結果，コンピュータを知らない人でもコンピュータを利用できるようになったのである．この点は，注目に値する．

マックの開発環境

統合環境はあたりまえ

プログラム開発がいかにもやりにくそうなマックであるが，実をいうとマックの世界ではエディタ，コンパイラ，リンカが１つになっているような，いわゆる統合環境が昔からあたりまえだったのである．

マックには，OSの上にファイル管理を行うFinderというソフトがある．このソフトを利用することで，マックをビジネスツールとして効率よく使うことができるのだが，残念なことにプログラム開発を手助けする環境を提供してはいなかった．結局，単独の開発ソフトの中で完結した開発環境を作り上げたほうが使い勝手がよいということになったのである．

マック発売当初は，Lisaを使ってプログラム開発が行われていたが，マック自身のハードウェア性能の向上とともに，マック上にも多くのプログラム開発ソフトが作られるようになってきた．プロのプログラマやハイエンド向けとしては，アップルが開発しているMPW（Macintosh Programmer's Workshop）が有名だが，この中では標準入出力やリダイレクトなどのUNIX的なOS処理ができるようになっており，マックのソフトとしてはある意味で異端者といえる．また，昔の製品では，MacCやLightSpeed C，Lightspeed Pascalあたりがやはり統合環境をもっていた．

現在は，MPWの系列とシマンテック社のTHINK C/Pascal以外はめだった開発ソフトがなくなってしまった．昔のほうが製品の種類は多かったのだ．これは，マックのアプリケーションがカスタマイズの域をはるかに越えた，プログラミング開発環境としての実力をもちはじめたことによる．たとえば，エンドユーザープログラミングのプラットフォームをめざすHyperCard，マルチメディアソフト開発ではマクロマインド社のDirector，データベースとなれば4th Dimmensionなどである．

ユーザーがアニメーションを作るのに，いちいちCだのPascalだのを勉強しなくても，Studio/1などのアニメーションソフトを使えばいいわけである．その結果，汎用開発ツールのフィールドが全体の広がりの中で少なくなってしまったといえるわけだ．

マックでの開発作業

必ずToolboxを利用

マックには，Toolboxというウインドウやメニュー処理などを行うシステムルーチンが集まったシステムソフトが存在し，これがMacOSの上位に位置している．マックのソフトは，必ずこのToolboxを経由して動作するように作らなければならない．標準形式のウインドウが嫌いだから，独自のウインドウを使うということもできるが，その場合もやはりToolboxでの規則に従ったウインドウ表示ルーチンを作成しないといけない．このToolboxの使い方を解説しているドキュメントが，Inside Macintoshである．日本語版もあり，現在５巻目まで発行されている．

Inside Macintoshの中には，たとえばウインドウを画面に出したいのならこのルーチンを使えだとか，ウインドウを動かすにはこうしろああしろということが書かれているわけだが，それらの情報量は膨大である．DOSでCのプログラミングをする場合，printf()でHello World!だとか自分の名前やらを画面に表示するという，かんたんかつ完結したプログラムを書くことができるので，初心者でも勉強をはじめたその日に，プログラミングをしたという実感を味わうことができるだろう．だが，マックではプログラムを完結させるために知っておかなければならないことがあまりに多く，しかもテキストベースではなくグラフィックスが主体になっているので，そのとっつきにくさはDOSの比ではない．

統合開発環境用語集
標準入出力　標準入出力ストリームの結合先．デフォルトではディスプレイとキーボードが指定されている．

しかし，グラフィックスインターフェースのライブラリが本体にもともと入っているわけで，GUI (Graphics User Interface) ソフトを作るとなると，場合によっては特殊なライブラリを作成する必要のあるDOSマシンに比べてはるかに効率的である．しかも，規定通りに作ったソフトはほかのアプリケーションとの親和性も高くなるのである．

Toolboxルーチンの使い方
高級言語から手続き名で指定する

では，このToolboxのルーチンを実際に使うにはどのようにすればよいかだが，マシン語レベルでは，68000のA-Trapという例外処理を開始する命令を利用している．「メニューを呼び出す処理」はどのトラップを使うということが規定されており，RAM内のある場所にあるトラップテーブルから実際のROMのルーチンや起動時に組み込まれたルーチンなどに飛ぶのである．

しかし，一般のアプリケーションを組む場合には，そのようなことを意識する必要はまったくない．Inside Macintoshには，Toolboxルーチンの使い方がPascalの呼び出し形式で記述されており，そのプロシージャ名や関数名をそのままリストの中に書き込めばよいだけだ(図1)．つまり，ライブラリや仕様拡張など処理系によって実現方法は変わるのだが，処理系の中でToolbox呼び出しを手続き名で指定できるようになっているのである．だから，トラップやテーブル，それに

Inside Macintosh / Window Manager

```
FUNCTION GetNewWindow(windowID: INTEGER;
                      wStorage: Ptr;
                      behind: WindowPtr) : WindowPtr;
```

Toolboxルーチンの呼び出すルーチン名や引数のわたし方がInside Macintoshで説明されている。

```
var
  windowID: INTEGER;
  wStorage: Ptr;
  behind:   WindowPtr;
  muWindow: WindowPtr;
begin
  ....
  muWindow:=GetNewWindow(windowID,wStorage,WindowPtr);
  ....
end
```

ソースコードではInside Macintoshに書かれた規則に従ってToolboxルーチンを呼び出す。

図1 Toolboxのシステムコールも高級言語のレベルで記述できる

引数渡しが，スタックだのレジスタだのといったマシン語レベルの処理が内部で行われてはいるが，プログラムとして書くときにはそういうチップそのものを連想させる概念はいっさい考えないで，CならCの関数の形式でそのままシステムルーチンが使えるわけだ．

システムコールといっても，MS-DOSのようにレジスタだとか割り込み番号をドキュメントで調べてというのとは大きく異なっている．

Cでは標準入出力もある
キャラクタベースのインターフェースも

マックの最近の開発環境では，キャラクタベースのインターフェースも用意されているのが普通になっている．たとえば，MPW CやTHINK Cでは，ANSI準拠のライブラリがついており，printf()やgetc()を使って文字ベースの入出力をもつソフトが記述できる．その際，処理系によってコンソール画面をシミュレートするウインドウが自動的に作られる．THINK Cの場合は，それがマルチウインドウのコンソールになったりする．

標準入出力も概念としては取り扱え，ちゃんとファイルをダイアログで指定できるなど，マック的な配慮もなされている．このため，Toolboxをまったく知らない，一般的なANSIのCしか知らないという人でもマックのプログラムは作れるのである．もちろん，コンパイルして単独のアプリケーションにして動かすこともできる．しかしながら，マックユーザーは「コンソール」というものをたいへんに嫌う傾向にあり，標準入出力を邪道と思う傾向があるので，要注意（？）だ．

THINK C
マルチウインドウで開発

THINK Cは，マックのソフトを作成するためのCの処理系である．プロ

統合開発環境用語集
標準補助入出力　標準補助入出力ストリームの結合先。デフォルトではRS-232 Cポートが指定されている。

グラミングに関わる作業は，おおむねこのソフトの中だけでできるので，統合環境といえるわけだ．以前はLightspeed Cといわれていた製品であるが，開発元がシマンテックに買収されるなどいろいろあって名称が一転二転し，現在は上記の名称になっている．実際にはアプリケーションは2本に分かれているのだが，あとで説明するように，デバッガはMultiFinder上で本体から起動されるようになっている．

THINK Cでは，1つのソースファイルが1つのウインドウになっている．それらソースはもちろんのこと，マルチウインドウで編集ができ，文字のカット&ペーストもできる（図2）．フォントやタブの設定はウインドウごとにできるようになっている．THINK Cでプログラムを行うときには，ソースに加えてプロジェクトファイルとリソースファイルが必要になる．

プロジェクトは，複数に分かれたソースファイルやライブラリを登録しておくウインドウである．さらに，プロジェクトファイルの中にコンパイルしたオブジェクトやライブラリのオブジェクトが蓄えられ，命令に応じてデバッグ可能な実行ファイルを生成する．

マックでプログラミングするときには，エディタがマルチウインドウであることを生かして，比較的短いソースファイルをたくさん作るようなやり方をする．1つのファイルに全ソースを入れてもいいのだが，モジュールごとにソースやヘッダなど多くのファイルに分けるのが一般的だ．そのたくさんあるソースを1つにまとめるのが，プロジェクトである．

ソース以外にリソースを使う

マック独自のデータ構造

リソースはマックに独特のデータ構造で，ウインドウやダイアログ，メニューなどはすべてリソースである．プログラムは，定義済みのリソースを呼び出すことでウインドウを表示したりダイアログを表示したりするのである．

たとえば，ウインドウならウインドウのタイプや大きさ，画面上の位置をリソースとしてデータ化し，リソースファイルに収めておく．リソースは，リソースエディタという別のユーティリティを使って作ることができる（図3）．リソースエディタでは，リソース

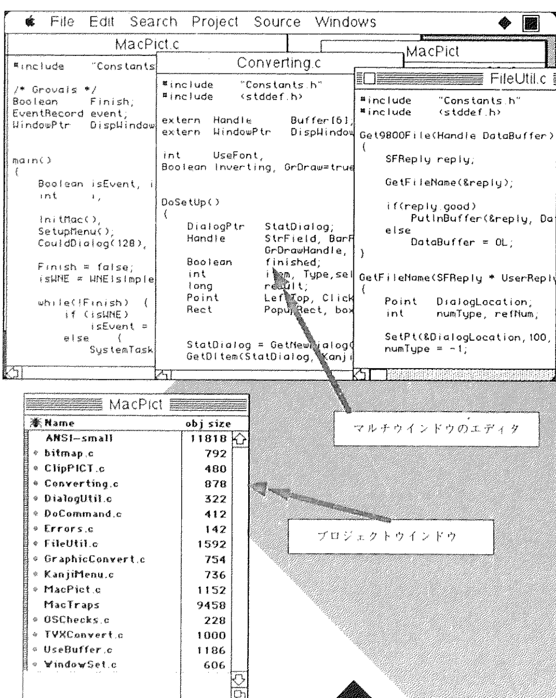


図2 THINK Cはソースファイルをマルチウインドウで編集できる

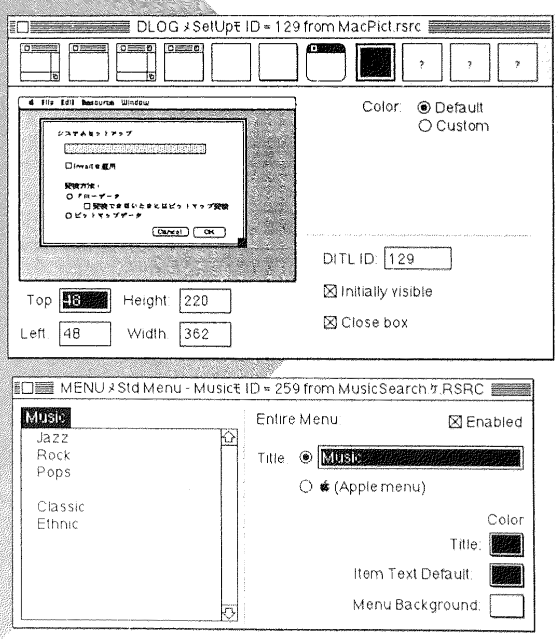


図3 リソース編集ソフトの代表はResEdit

統合開発環境用語集
プリプロセッサ　ソースファイルの一部分をある条件で加工したり，読み飛ばしたり，他のソースファイルを読み込むプログラム．

を実際に画面で見ながら作ることができる．リソースファイルには，たとえばウインドウならウインドウのグラフィックス情報がビットマップなどで入るのではなく，構造体，すなわちデータ列として保存される．この構造体は，システムで定義されており，ウインドウの大きさや種類など管理データの集まりである．

リソースエディタの中では，図3に示したアップルによって開発されているResEditが有名で，THINK CにはResEdit Ver.1.2が添付されている．しかし，最近，ResEdit Ver.2.1というのがアップルから供給されるようになり，機能も大幅にアップした．

デバッグ

MultiFinder上で使うTHINK Cのデバッガ

ソースレベルデバッガは，MultiFinder上でのみ利用できる．デバッガを使用するように指定して実行すると，図4のようにデバッガに入った状態でユーザーからのアクションを待つ．画面の下側の2つのウインドウがデバッガで，左はソースコードを右は変数をモニタする．このときには，THINK Cの本体に加え，デバッガ，それからコンパイルしたプログラムがそれぞれMultiFinder上で起動しているのである．つまり，作成したプログラムは実際にメモリ上に展開されているわけだ．

デバッガでは，ステップ動作やブレーク，条件つきブレークなどができ，Cのソースコードレベルでのデバッグができる．変数を見るウインドウでは，

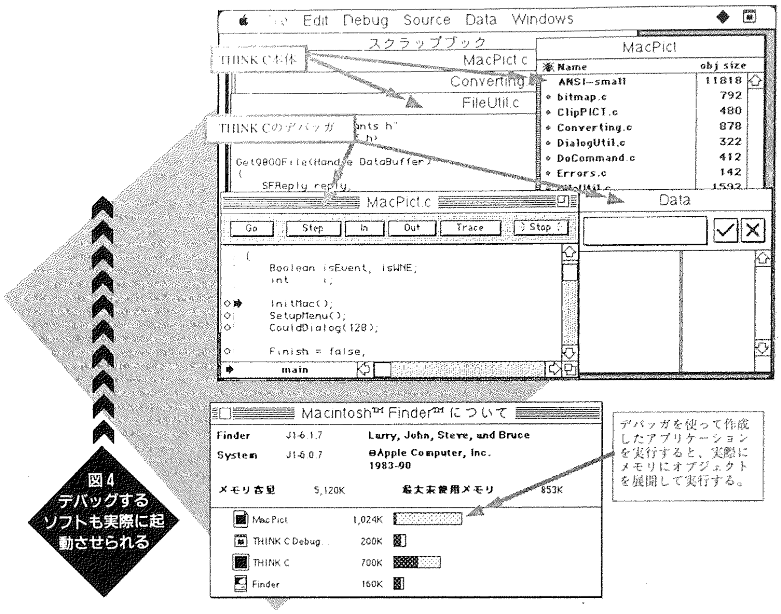


図4 デバッグするソフトも実際に起動させられる

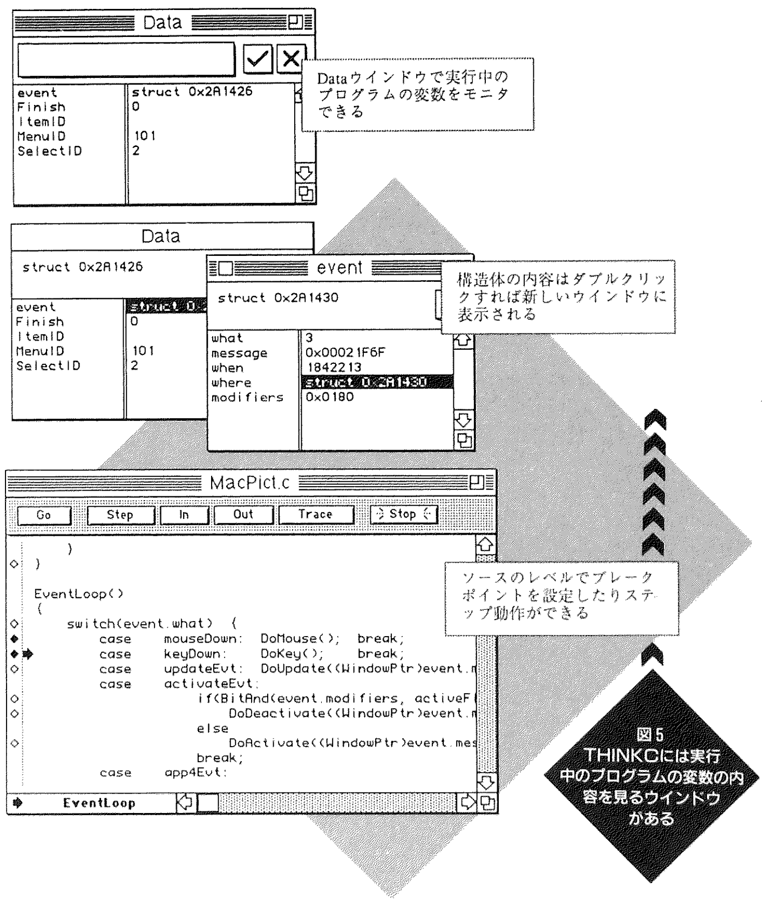


図5 THINK Cには実行中のプログラムの変数の内容を見るウインドウがある

図5にあるように構造体なら，そのメンバなども見ることができる．また，さらに低レベルのマシン語レベルのデバッガMacsBugもあるが，これは別途入手しなければならない．

THINK Cのそのほかの仕様をかいつまんで紹介しておこう．まず，アセンブラももっており，ソースコード

雰囲気はまるでUNIX

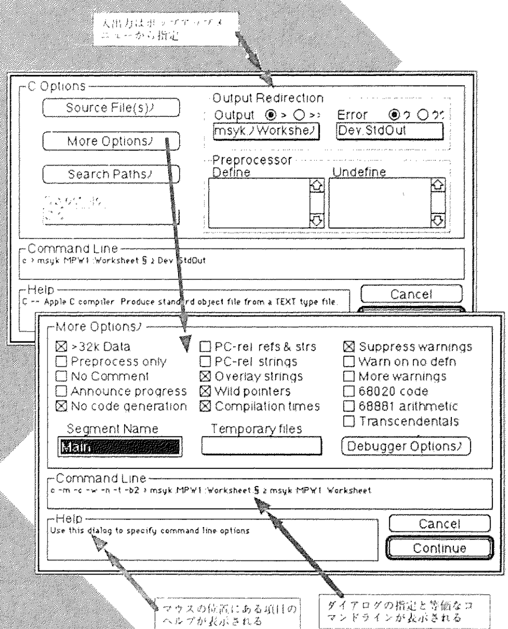
MPW（Macintosh Programmer's Workshop）

MPWは，本文中にも書いたが，マックのソフトにしては一風変わっている．もちろん，MPWも普通のマックと同様Finderからダブルクリックで動きだすのだが，一度MPWの世界に入ってしまうと，マウスやプルダウンメニューなどのマックらしい操作と，UNIXやDOSなどのようなコマンドラインで引数やスイッチを与える記述が渾然一体となってしまうのである．このため，マックのOSの上で動いてはいるものの，新たなOS環境を提供しているものといったほうがよいだろう．

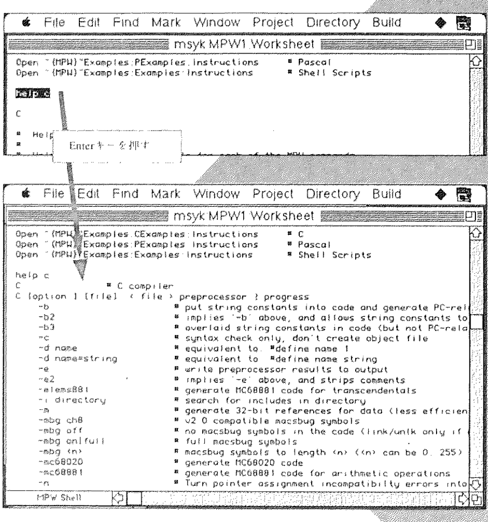
MPWには，UNIXなどと同じようなシェルといわれる核の部分がある．このシェルに組み込まれたコマンドや，コマンドの記述を収めたスクリプト，MPWのシェル上でだけ機能するツール，一般のマックのアプリケーションなどをシェルから起動し，利用できる．このあたりの雰囲気はまったくUNIXである．

シェルは図Aのように，一見するとマックの普通のワープロのようなウインドウ形式をもっている．だが，MPWではウインドウ内でコマンド記述されている文字列をセレクトして，Enterキーを押すことでコマンドを与える（というよりシェルにメッセージを送るといったほうがいい）．すると，そのセレクト範囲にコマンド実行による出力が書き込まれるのである．このウインドウは，ソースのウインドウにもなり得るのだが，コマンドを与えるコンソールとしても使用できる．

DOSのバッチファイルに相当するスクリプトやツールでは，標準入出力や診断出力，コマンドラインでのオプションスイッチなどがあり，記述そのものはUNIXライクである．だが，このファイル指定やオプション指定は，図Bのようにダイアログボックスを表示させてもできるので，それらのコマンド入力をマックライクに進めることもできるわけだ．



図B　コマンドラインの指定をマックライクにできる



図A　ウインドウの中の文字列をコマンドとして送り出す

MPW上には，CをはじめC++やObject Pascalなどさまざまな言語処理系がある．これらのコンパイラは，シェルからコマンドとして呼び出すようになっている．また，オブジェクト指向プログラミング用にはMacAppというライブラリがあり，これは歴史も古くマックのアプリケーションプログラムをオブジェクト指向で作る際の標準的なプラットフォームになっている．また，ソースレベルのデバッガやユーティリティも各種備えられている．

MPWは，アップルの開発者向けの組織であるAPDA（Apple Programmers and Developpers Association）から入手できるが，シェルからコンパイラなど一式がCD-ROMになっているE.T.O.と呼ばれる製品が最近発売された．もちろん，必要なものだけを買うのもよいのだが，バラで買うよりCD-ROMを1枚買うほうがお買い得なのはもちろん，ディスクの洪水も防ぐことができる．MPWのシェルとCコンパイラのマニュアルは，アスキーから日本語に翻訳されたものが出版されている．

統合開発環境用語集
ブレークポイント　デバッガのターゲットプログラム中で，あらかじめ動作を停止するポイント（アドレス・ラベル）のこと．

の中でニーモニックで記述することができる。アセンブラおよびコンパイラのいずれも，68000だけでなく68020や68881のコードを生成することができる。図6のように，ソースウインドウからヘッダファイルをワンタッチで呼び出すことができるようになっている。また，検索や置換の処理も複数のファイルにわたって行うことができ，正規表現も使うことができる。

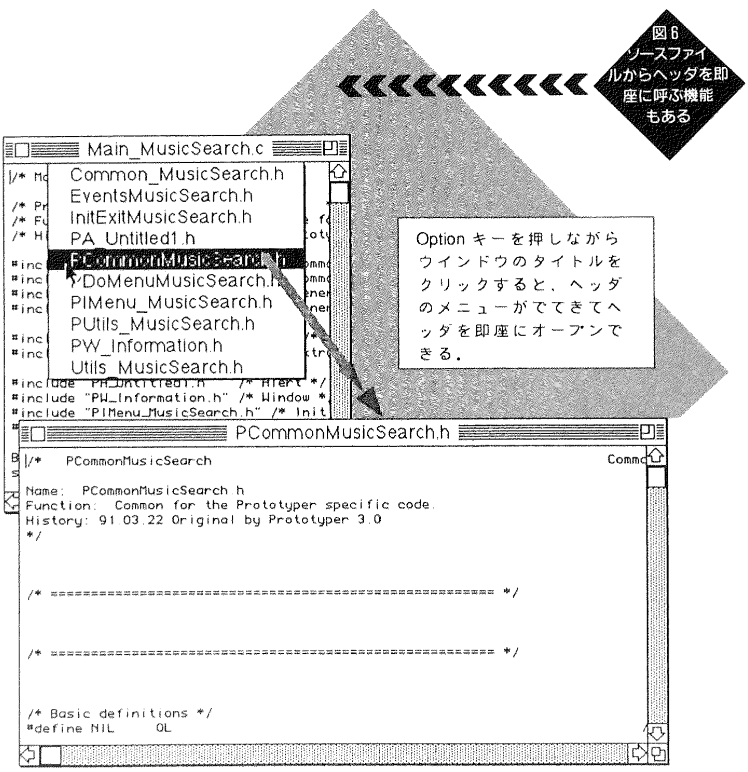
THINK Cでは，オブジェクト指向プログラミングもサポートしているが，言語仕様としてオブジェクトが扱えるというレベルであり，ブラウザやデバッグでのオブジェクト指向プログラミング支援はほとんどないといってよい。だが，TCL(THINK Class Library)というクラス定義ライブラリがあるので，オブジェクト指向プログラミングによってマック用のアプリケーションソフトを作成することができる。

オブジェクトライブラリでは，ウインドウやメニュー，ボタン，テキストフィールド，スクロールバーなど，いわゆる画面上に表示されるビジュアルオブジェクトに加え，ドキュメントクラスとしてのファイル管理やクリップボードの管理，リソースの処理など，アプリケーションを作成するのに必要なクラスはほとんどそろっている。ただ，マニュアルに仕様が書かれていないクラスもあり，まだまだ発展途上のライブラリのようだ。

THINK Cで日本語を処理するプログラムを作成することはできるが，リストの中に漢字を入力することは実質的には不可能といえる。リストファイルのフォントを日本語フォントにすれば，画面上には文字化けせずに表示されるが，漢字が入るとカーソルがマウスでクリックした位置とずれたりしてうまく編集ができないのである。したがって，ソースに漢字を入れることはできないに等しい。ほかのソフトで文字を編集して読み込んでもよいのだが，そういう使い方は開発環境とはいわないだろう。プログラムで表示する文字が漢字ならば，文字列をリソースとして扱うようにし，リソースエディタで漢字文字列を入力するのがよい。もっとも，マックでは英語であっても文字列はリストに含めずにリソースにすることが多いのである。ただ，リストに漢字でコメントを入れたいとなると，THINK Cの環境をあきらめ，ワープロでリストを読み込んでコメントをつけることになる。

Prototyper
ユーザーインターフェースを構築・検証する

図6 ソースファイルからヘッダを即座に呼ぶ機能もある



Prototyperは，かんたんにいえばコードジェネレータなのだが，マック用ソフトのインタープリタ的な要素も重要であり，ただのジェネレータとはいえない。

Prototyperでは，まるでグラフィックスソフトのようにウインドウやダイアログ，メニューを設計することができる（図7）。そして，「RUN」という命令で，ほんとうにRUNするのである（図8）。これだけを見ると，ウインドウあるいはメニューの作成ツールのように思うかもしれないが，たとえばウインドウ内のあるボタンを押したら，どのような処理を行うのかまでをPrototyperで設定できる。つまり，OKボタンを押せば新たにウインドウを開いて，メニューのいくつかのアイ

統合開発環境用語集
ホットキー　ショートカットキーと同じ意味で用いられる。コマンドをメニューから選択するのではなく，ひとつまたは複数のキー入力で行う。

テムを選択できないようにして……などという一連の手順を設定するわけだ。これは，まさにフォーム作成の領域を越えてプログラミングの世界に入っているといえる。

マックをはじめとするGUIのソフトでは，うまくすれば内部処理とユーザーインターフェースを分離できる。Prototyperは，その性質を巧みに利用したユーザーインターフェース構築ツールだ。RUNすれば，メニュー選択やウインドウ表示などが設定されたとおりに行えるので，これを利用してソフトの実際の使い勝手を検証することができる。

ウインドウのデザインが決まれば，同じ動きをするソースを生成してくれる。THINK C/Pascalはもちろん，MPW C/Pascalなどのソース生成も行う。生成されたソースは，図9のようにTHINK Cでプロジェクトを新規に作成して読み込めば，コンパイルされてちゃんと動き出す。コード生成も指定したファイルに対してだけできるなど，フィードバックのあるような開発でも使えるように配慮されている。

Prototyperは，コーディングに入るまでの作業を支援するものだが，Prototyperが生成したコードを見ることで，プログラミングの勉強になることもある。つまり，サンプルコードを作るシステムとしても利用できる。

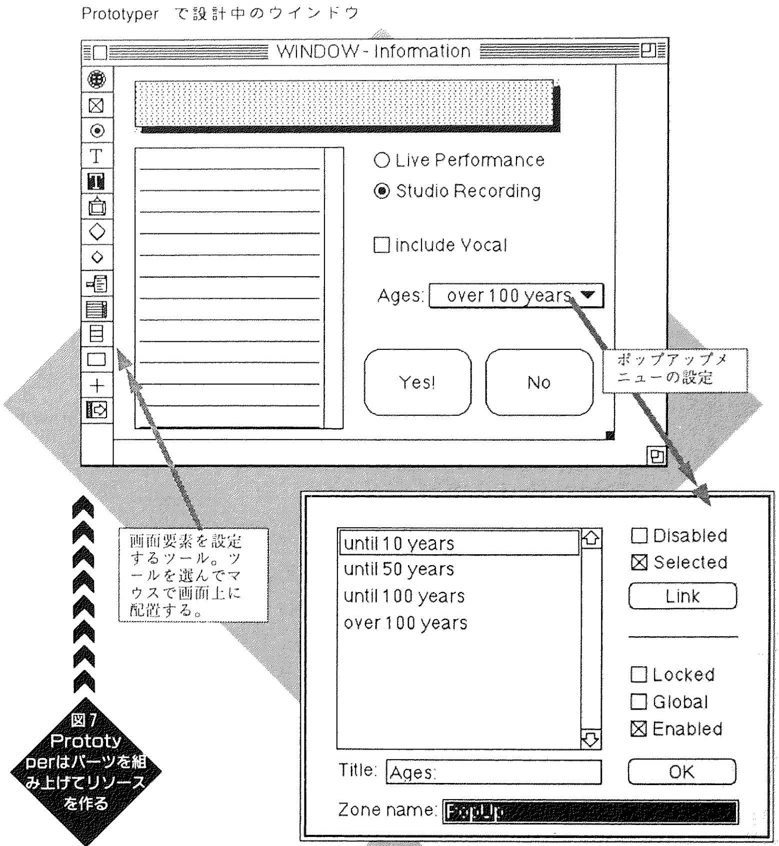


図7 Prototyperはパーツを組み上げてリソースを作る

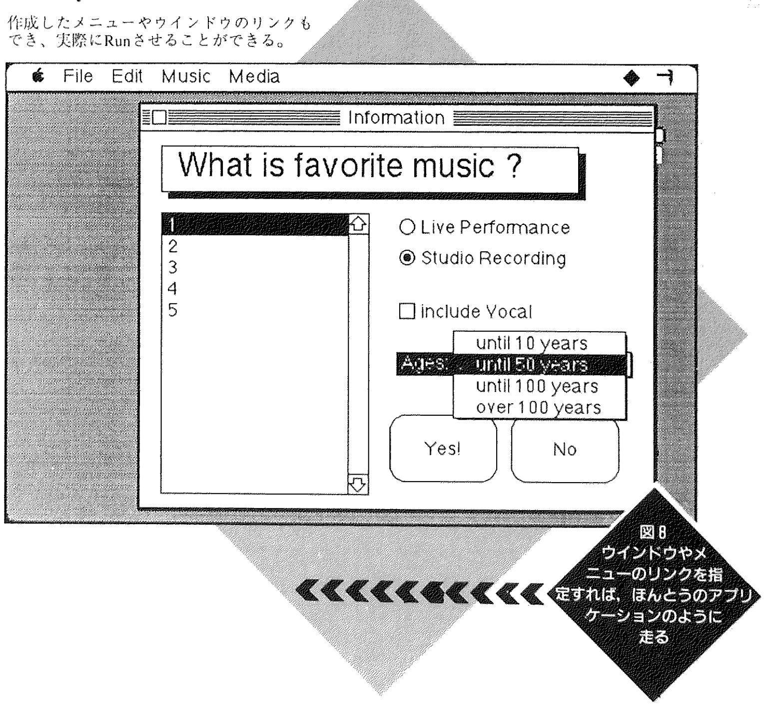


図8 ウインドウやメニューのリンクを指定すれば，ほんとうのアプリケーションのように走る

Online Companion

Inside Macintoshのヘルプを呼ぶ

Inside Macintoshは，マックでプログラムを行うためには必ずもっていないといけないドキュメントなのだが，肝心な部分，たとえばNewWindow関数の引数はどんなのだったかなどを調べるのに，いちいちぶ厚い本を開くのもたいへんである。Online Com-

統合開発環境用語集
ホットスポット　ヘルプテキストなどでクロスリファレンスが配置された単語や文章。選択することでクロスリファレンスがアクティブになる。

panionは，そういったInside Macintoshで定義されているルーチンやマクロ定数などのオンラインヘルプを行うソフトである(図10)．どんなソフトが動いていても，［コマンドキー］＋［`］で画面上に表示される．

複数のソフトを組み合わせる

CD-ROMは不可欠なツール

THINK Cなど，統合環境と分類される開発ソフトを利用するのも1つの手段だが，それに加えてPrototyperなどさまざまなソフトを活用して開発を進めることで，より高いレベルの統合環境を構築できる．もちろん，Prototyperを使ったからといって，そのコードを絶対に使わないといけないというわけではない．たとえば，Prototyperで作ったダイアログリソースだけを使って，THINK Cで実際のコーディングをするという用途も考えられる．また，プロトタイプツールとしてはHyperCardも見逃せない．スタックである程度作り，それを汎用開発言語で完成させるというやりかたも考えられる．

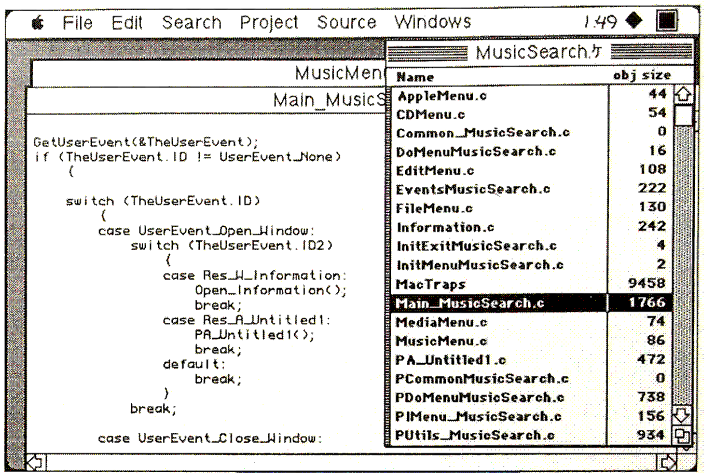


図9 Prototyperの生成コードはTHINK Cでコンパイルできる

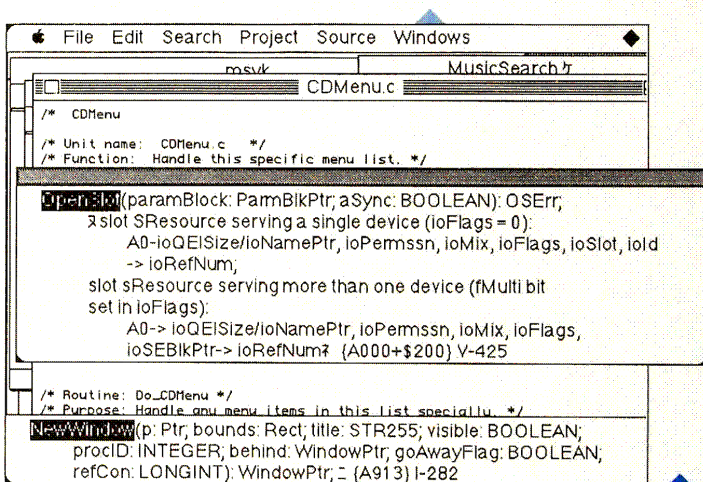


図10 画面の下にInside Macintoshのダイジェストが表示される

マックのソフト環境を考えるときに忘れられないのがCD-ROMだ．CD-ROMドライブは，マックでソフトを作っているユーザーには不可欠になってきている．MPWはフルセットをCD-ROMで供給しており，フロッピーディスクで買うよりも安い値段になっている．それに，アップル社が発行している開発情報の季刊誌「d e v e l o p」には毎号CD-ROMがついており，雑誌で紹介しているリストはもちろん，システムや開発ツールの最新版やベータ版などがその中に入っている．

Apple Developer Groupも「Developer CD」という名前でシステムやツールなどを供給しており，現在6巻までリリースされている．この中には，HyperCardのスタックになっているTechnical Notes (マックの技術情報で数か月ごとに発行されている)や，InsideMacintoshの追加情報など，必要な情報が電子化されて収められている．また，これからリリースされる新システムのSystem7.0も，ベータ版はCD-ROMで開発者に配付されているという．これらの情報を目にしておかないと最新の状況に乗り遅れるわけで，マックでのソフト開発にはCD-ROMドライブは必須になっている．

統合開発環境用語集
マウス　統合開発環境ではコマンドの選択やカーソルの移動などをキーボード以外にもマウスを使ってメニューから選択できる．

これからの統合環境

Windows上の統合環境プログラミング



Turbo Pascal for Windowsは，ボーランド社がBorland C++に続いて発表したMS-Windows対応のコンパイラである．Turbo Pascal for WindowsとBorland C++とのもっとも大きな違いは，Borland C++がDOS上で動作するIDEしか備えておらず，Windows用アプリケーションの開発をDOS上で行わなければならないのに対して，Turbo Pascal for WindowsはWindows IDEと呼ばれるWindows上で動作する統合環境を備え，Windows上でアプリケーション開発が行える点である．

Windows用アプリケーションの開発は難しいとよくいわれるが，これは500以上にのぼるWindows API (Application Program Interface)を理解するのにたいへんな時間が必要なためだ．Turbo Pascal for Windowsでは，この問題を解決するためにObjectWindowsと呼ばれるオブジェクトライブラリを備えている．ObjectWindowsをアプリケーションのフレームワークとして利用することで，C言語で記述すると100行を越える最小の機能をもったWindowsアプリケーションをわずか10数行で記述することが可能になる．

Turbo Pascal 6.0にもTurbo Visionと呼ばれるユーザーインターフェースクラスライブラリが付属しているが，残念ながらObjectWindowsとの互換性はない．

Turbo Pascal for Windowsにはこのほかに，Whitewater Resource ToolkitとTurbo Debugger for Windows，Turbo Help Hypertext Systemが含まれている．これらによって，SDK(Software Development Kit) なしでWindowsアプリケーションの開発が可能になる．

Turbo Pascal for Windowsは，いまのところもっとも安価に，それでいて本格的なWindowsアプリケーションが開発できる製品といえる．まさに“Pascalのボーランド”の面目躍如といったところだろう．

●製品名：Turbo Pascal for Windows
●価格：249.95ドル
●問い合わせ先：（株）ボーランドジャパン
●電話：03-3486-1400

```
program Hello;
uses WObjects, WinTypes, WinProcs;

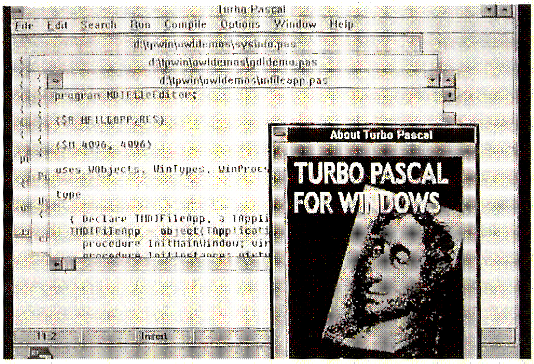
type
  THelloApp = object(TApplication)
    procedure InitMainWindow; virtual;
  end;

  THelloApp.InitMainWindow;
  begin
    MainWindow := New(PWindow,
      Init(nil,'Hello, Windows'));
  end;

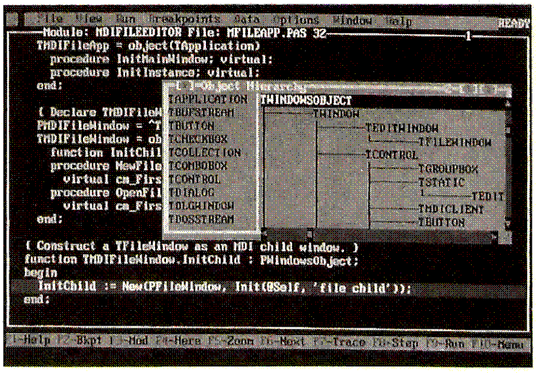
var HelloApp:THelloApp;

begin
  HelloApp.Init('HelloApp');
  HelloApp.Run;
  HelloApp.Done;
end.
```

Hellow Windows！プログラム



Windows IDE



Turbo Debugger for Windows

統合開発環境用語集
マルチウィンドウ　統合開発環境では，ソースファイルを表示するウィンドウ以外にエラーメッセージやヘルプなどのウィンドウを同時にオープンできる．



Smalltalk/V for Windowsは，米国DIGITALK社が発売していたSmalltalk/VのWindows対応版である．Smalltalk/Vは，DOS上で動作する安価なSmalltalkとして人気があり，またOS/2のプレゼンテーションマネージャに対応したSmalltalk/V PMは，Smalltalk初のコンパイラとして高い評価を受けている．今回発売になったSmalltalk/V for Windowsは，このSmalltalk/V PMと高い互換性をもっているといわれ，WindowsとPMのプログラミングプラットフォームとして注目されている．当然ながら，すべての開発をWindows中で行え，SDKなどは不要である．

Smalltalk/V for Windowsは，プログラミングを支援するための多くのウインドウをもっている．この中でプログラミングの中心になるのが，Class Hierarchy Browserである．このウインドウ中には，Smalltalk/Vがもっているクラスやメソッドが示される．Smalltalk/Vにおけるプログラミングは，これらのメソッドを変更したり，新しく作成したメソッドを付け加えることで行われる．

Inspectorは低レベルのデバッグを支援するウインドウで，オブジェクトの内容を調べたり，変更するのに使われる．

Walkbackウインドウは，エラーが生じたときに自動的に画面に現れ，エラーに関する情報を表示する．Walkbackウインドウが現れたときは，その情報を元にプログラムを修正するか，さらに詳しく調べるためにデバッガを起動することになる．

Debuggerウインドウは，4つのペイン（タイルドウインドウ）に分かれている．左上のペインにはWalkbackウインドウと同じ情報が示され，ここで行を選択するとその行に関する情報がほかのペインに示される．ウィンドウの下に示されているペインでは，Class Hierarchy Browserと同じようにメソッドの編集が行える．

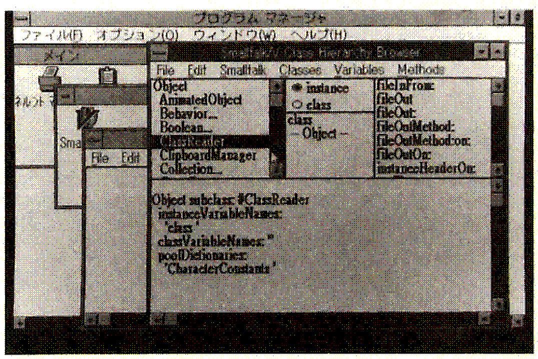
このほかにも，オブジェクトが送るメッセージを表示させることも可能である．

Smalltalkは，言語仕様から開発環境まで，すべてがオブジェクト指向で

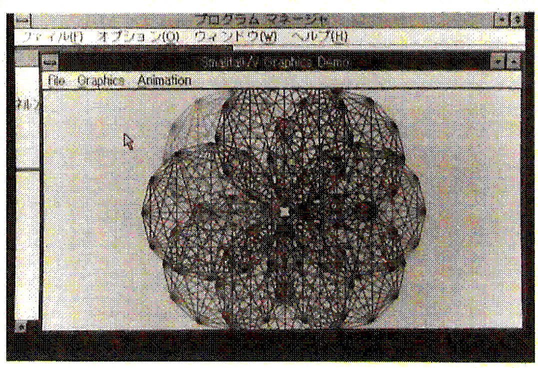


統一されているため，C言語などに慣れたプログラマにはちょっととっつきにくいかもしれないが，Windowsのようなオブジェクト指向にマッチする環境では強力な武器になることだろう．

●製品名：Smalltalk/V for Windows
●価格：85,000円
●問い合わせ先：
（株）サザンパシフィック
●電話：045-314-9514



Class Hierarchy Browser



グラフィックデモ



Actor 3.0は，米国Whitewater社が発売しているWindows 3.0用のアプリケーション開発システムである．Smalltalk/V for Windowsと同じくすべての開発をWindows中で行うことができる．SDKが不要なことはいうまでもない．

C言語などでは，一般にエディット，コンパイル，リンク，プログラムの実行，デバッグという流れで開発が行われるが，Actorでは，クラスライブラリのブラウジング，新しいクラスの作成，メソッドの記述，プログラムの実行，デバッグ，Seal-Off操作，リソースコンパイラによる最終的な実行形の作成という流れでプログラム開発が行われる．

プログラマが，１つメソッドを記述するたびにActorはメソッドをコンパイルしてクラスに付加していく．同時にActorは文法チェックを行い，単純なタイプミスなどを拾い出してくれる．

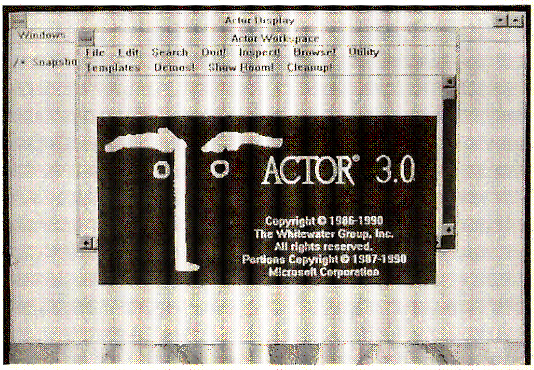
メソッドを記述し終えたら，Workspace上で今書いたプログラムを実行させてテストを行い，エラーがでたら，デバッガを起動してコードを修正する．Actorのデバッガでは，オブジェクト間のメッセージやメッセージ中で使われている変数などを調べることが可能である．

Actorは，コンパイル後のコードをイメージファイルとしてセーブしているが，この中にはActorのツールや余分なクラスが含まれているので，最終的なアプリケーションを作成する際には，これらを切り離してアプリケーションに必要なコードのみのイメージファイルを作成する．この操作をSeal-Offと呼んでいる．

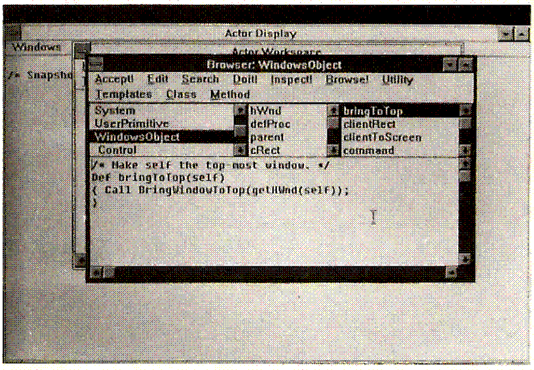
次にACTOR.EXEをリソースコンパイラの存在するディレクトリにコピーし，Seal-Offされたイメージファイルと同じ名前にリネーム（ただし，拡張子は．EXEとする）し，リソースコンパイラを用いてリソースファイルを付加すれば，アプリケーションが完成する．

Actorを支援するツールとして，Whitewater社はWhitewater Resource Toolkit（WRT）を発売している．これは，ダイアログボックス，ビットマップ，アイコン，カーソルなどのリソースを作成するためのツールで，あらゆるリソースを対話的に画面上で確認しながら作成することを可能にしている．

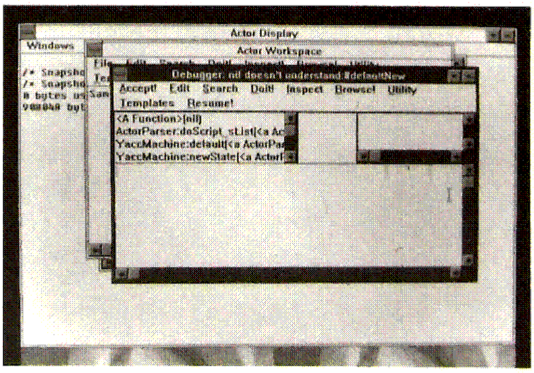
●製品名：Actor 3.0
●価格：188,000円
●問い合わせ先：（有）データポット
●電話：052-798-4154



起動画面



Browser



Debugger

統合開発環境用語集
予約語　識別子として使えないキーワード．ANSI Cでは32語が予約語になっている．

統合環境を備えた言語製品紹介

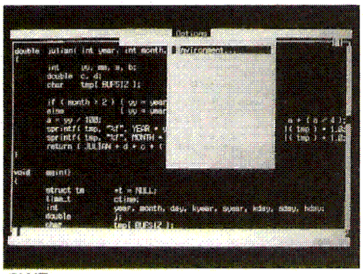
Microsoft C Professional Development System Version 6.0

MS-Cは，従来これといった統合開発環境をもたないオーソドックスなコンパイラだったが，バージョン6.0から PWB (Programmer's Work Bench) と呼ばれる強力な統合開発環境をもつようになった.

PWBは, QuickCなどのように1つのプログラムにエディタ，リンカ，コンパイラ，デバッガを含むのではなく，それぞれ独立したエディタ，リンカ，NMAKE, CodeView, コンパイラなどのプログラミングツールを統合して動作させることによってプログラミング環境を作り上げている.

PWBに付属しているエディタは，マイクロソフトエディタを機能強化したもので，マルチウインドウ，マルチファイル，マウスによる操作が可能となっている. また, PWBには開発中のプログラムを構造解析するソースブラウザ機能やオンラインリファレンス (Microsoft C Advisor) も含まれている.

PWBはMS-Cだけでなく，今後発売されるマイクロソフト社の言語製品の多くに標準添付されるということである.


PWB

●製品名：Microsoft C Professional Development System Version 6.0
●価格：98,000円
●対応機種：NEC PC-9800シリーズ
●問い合わせ先：マイクロソフト(株)
●電話：03-3363-1201

MS QuickC 2.0

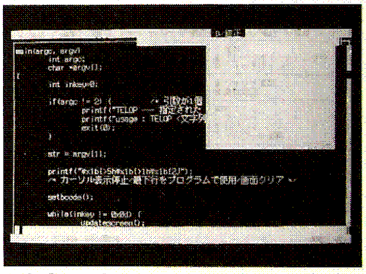
QuickC 2.0は，MS-C 5.1と互換性のあるC言語入門者用のCコンパイラである.

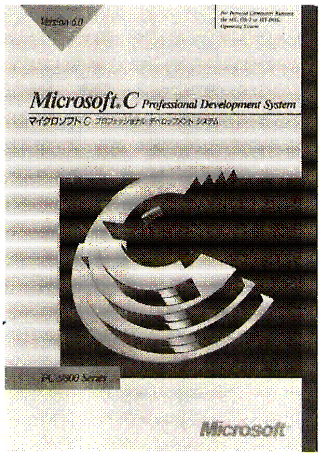
QuickC 2.0には，エディタ，リンカ，コンパイラ，デバッガを統合した統合環境版とそれぞれを独立させたツールキットの両方が含まれている. 統合環境版においてもすべてのメモリモデルのプログラム作成が可能で，これといった制限はない. また，インラインアセンブラ機能をもっているので，C言語のプログラム中にアセンブリコードを記述することができる.

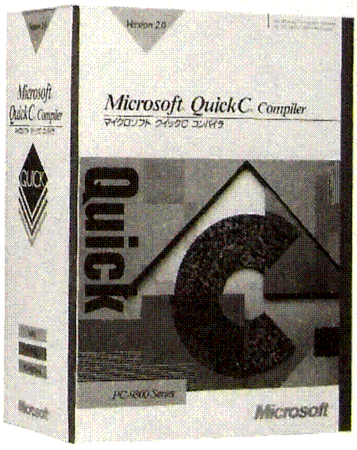
QuickC 2.0には，オンラインヘルプとしてQCアドバイザが備えられ，ハイパーリンク形式で次々に必要な説明を表示させることが可能となっている. また, 付属のHELPMAKEを利用すると，ユーザー独自のヘルプを作成することもできる.

さらにQuickC 2.0のパッケージには，C言語の初心者向け入門書として「C for Yourself」が付属している.

●製品名：Microsoft QuickC 2.0
●価格：20,000円
●対応機種：NEC PC-9800シリーズ
●問い合わせ先：マイクロソフト(株)
●電話：03-3363-1201


MS-QuickC





MS-QuickBASIC 4.5

QuickBASIC 4.5の統合開発環境は，エディタ，インタープリタ，デバッガ，リンカ，コンパイラを統合したものである．また，オンラインヘルプとしてハイパーリンクを実現したQBアドバイザも装備している．

QuickBASICのプログラミングは，従来のBASICプログラムとは異なり，PascalやC言語と同様の構造化が可能で，行番号は不要，goto文を使用しないプログラムが作成できる．さらに，プロシージャの概念が導入され，ローカル変数の使用も可能である．

また，BIOSの割り込みを利用できるInterruptプロシージャが付属しているため，マシンに近い低レベルなプログラムも作成できる．

QuickBASIC 4.5は，ランタイムライブラリ別体系のexeファイルだけでなく，独立型のexeファイルの作成が可能で，さらにC言語やアセンブラ，FORTRANなどの言語とのミクスドランゲージプログラミングにも対応している．

●製品名：Microsoft QuickBASIC 4.5
●価格：20,000円
●対応機種：NEC PC-9800シリーズ
●問い合わせ先：マイクロソフト(株)
●電話：03-3363-1201



Turbo C++

Turbo C++は，AT&T C++ Ver.2.0に準拠したC++コンパイラである．

Turbo C++の統合環境は，従来のTurboシリーズの統合環境をさらに発展させたもので，プログラマーズプラットフォーム（Programmer's Platform）と呼ばれている．エディタは，マルチファイル，マルチウインドウを実現している．しかも，このウインドウはオーバーラッピングウインドウで，そのサイズや位置を自由に設定することができる．また，ファイルやウインドウの数には制限がなく，取り扱えるファイルの大きさはディスク容量のみに制限される．

マクロ言語TEML（Turbo Editor Macro Language）を用いてマクロやキー定義をし，TEMC(Turbo Editor Macro Compiler）を通して独自のプラットフォームの作成ができる．

トランスファ機能を用いると，Turbo AssemblerやTurbo Debuggerなどの外部プログラムを統合環境から呼び出すことができるようになる．また，Turbo Assemblerやgrepなどの出力を利用したタグジャンプも可能である．

進化したオーバーレイマネージャであるVROOMM(Virtual Runtime Object-Oriented Memory Manager）によって，プログラマはとくに意識することなく効率のいいオーバーレイを実現することができる．

●製品名：Turbo C++ 1.0／同Professional
●価格：39,800円／59,800円
●対応機種：NEC PC-9800シリーズ
●問い合わせ先：(株）ボーランドジャパン
●電話：03-3486-1400

Turbo Pascal 6.0

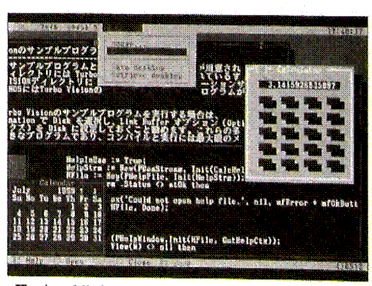
Turbo Pascal 6.0には，オブジェクト指向のユーザーインターフェースクラスライブラリであるTurbo Visionが付属している．このクラスライブラリを利用することで，ユーザーはかんたんに位置・サイズを変更可能なオーバーラッピングウインドウやプルダウンメニュー，ダイアログボックスなどを装備したイベントドリブンなアプリケーションを構築することができるようになる．

Turbo Pascal 6.0に付属している統合環境がこのTurbo Visionを用いて作られていることからも，その実用性の高さがうかがえる．

また，統合環境に備わっているデバッガには条件ブレークポイント機能やレジスタウインドウなどが追加された．

従来，Turbo Pascalのインラインアセンブリ機能は外部のTurbo Assemblerを呼び出すことで実現されていたが，6.0から組み込みアセンブラが使用できるようになった．

また，Turbo Pascal 6.0 Professionalには，プロテクトメモリを利用



Turbo Visionのデモ

統合開発環境用語集
リダイレクト　標準入力や標準出力先を変更すること．普通は標準入力はキーボード，標準出力はディスプレイが指定されている．

したコマンドライン版コンパイラが付属し，巨大なアプリケーションもコンパイル可能となっている．

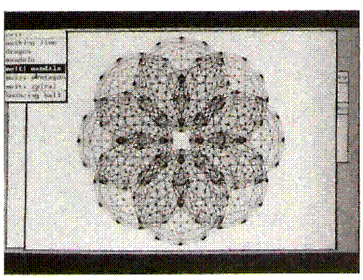
●製品名：Turbo Pascal 6.0／同Professional
●価格：29,800円／49,800円
●対応機種：NEC PC-9800シリーズ
●問い合わせ先：（株）ボーランドジャパン
●電話：03-3486-1400

Smalltalk/V

Smalltalk/Vはオブジェクト指向という考え方をもとに，エディタ，デバッガ，ライブラリアン，ファイルマネージャ，ウインドウシステム，プログラミング言語などをすべて統一した環境を提供している．

言語仕様からユーザーインターフェースまでがオブジェクト指向で統一されているため，プログラミング環境のユーザーインターフェースをかんたんに自分のアプリケーションに取り込んで利用することが可能となっている．

Smalltalk/Vにおける実際のプログラミングはメソッドの作成という形で，クラス階層ブラウザ（Class Hierarchy Browser）と呼ばれる特別なウインドウを用いて行われる．クラス階層ブラウザ中では，既存のクラスやメソッドの参照，変更，新しいメソッドの作成が行える．


Smalltalk/Vのデモ

Smalltalk/Vでは，アセンブラやC言語で書かれたプログラムを呼べるので，ハードウェアなどを直接扱うプログラムなどの作成も行える．

●製品名：Smalltalk/V
●価格：49,800円
●対応機種：NEC PC-9800シリーズ
●問い合わせ先：（株）サザンパシフィック
●電話：045-314-9514

THINK C

THINK Cは，Macintoshのすぐれたユーザーインターフェースをそのままプログラミングに生かした統合環境をもつＣ言語コンパイラである．THINK Cの統合環境には，高速コンパイラ，リンカ，ソースレベルデバッガなどが含まれている．

THINK Cでは，手続き型とオブジェクト指向型の２つのプログラミングから１つを選ぶことができる．このオブジェクト拡張機能は，C++に基づくもので，各クラスはC++の上位互換となっている．

THINK Cに付属しているTHINK Class Libraryは，標準Macintoshインターフェースを実現する強力なもので，フローティングウインドウやティアオフメニューなどの特殊項目のほかにウインドウ，メニュー，コントロール，パレットなど，通常のMacintoshインターフェースを構成するすべてのものが含まれている．

ソースレベルデバッガは完全にオブジェクト指向プログラミングに対応しており，メソッドの追跡，インスタンス変数の検査・修正などが行える．

●製品名：THINK C
●価格：48,000円
●対応機種：Macintosh
●問い合わせ先：(株)システムソフト営業企画課
●電話：092-722-4857

THINK Pascal

THINK Pascalは，超高速コンパイラ，高速リンカ，ソースレベルデバッガなどを統合したオブジェクト指向のPascalコンパイラである．

THINK Pascalでは，アプリケーションのほかにマルチセグメントのデスクアクセサリ，デバイスドライバ，そしてINIT，cdev，HyperCardのXCMDとXFCNを含む各種のコードリソースを作成することができる．

THINK Pascalに付属しているTHINKクラスライブラリは，標準的なMacintoshユーザーインターフェースをインプリメントするためのビルディングブロックを提供する．このTHINKクラスライブラリとともにApple社のクラスライブラリであるMacAppをサポートしているので，オブジェクト指向プログラミングをフル活用することができる．

また，クラスブラウザをエディタとともに用いることで，開発中のプログラムの全クラスの表示，定義の確認，クラスおよびメソッドを検索・編集することが可能である．

●製品名：THINK Pascal
●価格：48,000円
●対応機種：Macintosh
●問い合わせ先：(株)システムソフト営業企画課
●電話：092-722-4857

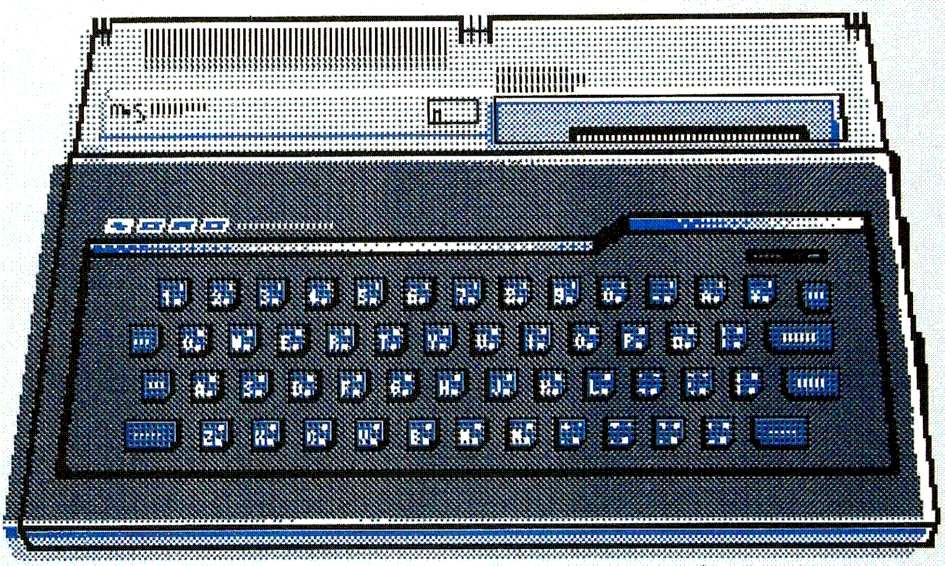
パソコン博物館 VI

ベンチャービジネス，希望の星

SORD M Series

ソード社 SORD Mシリーズ

井戸禎光



ファミコンなどのゲーム機の元祖 SORD M5

Illustration
TOSHIHITO ITO

今回は，日本にもパソコン界の天才が現れたということについて考えてみたい．1970年代は，マイクロプロセッサという新しい技術の登場の機会をとらえて，パソコン産業が登場するのであるが，日本と米国では企業の対応のしかたが異なっていた．米国では，アップル社をはじめとするベンチャービジネスがチャンスをとらえたのに対して，日本では今回登場するSORD社しかベンチャービジネスが登場しなかったことは奇異に感じる．果たして，何がその理由であったのであろうか．

一般にベンチャービジネスの特徴は，即断，即決，即実行，先端技術の取り込み，ユーザーニーズの取り込みなど，小回りがきくことであろう．

さて，SORD社は，1970年3月に椎名堯慶氏が母親からお金を出資してもらい，数人の学生時代の友人とはじめたコンピュータ会社である．1974年には，米国でアルテア8800が出ているが，このころすでにSMP80/20を出しており，1977年には米国における第二期黄金時代を飾るApple II，PET2000，TRS-80と期を同じくして，M200シリーズという実用機を売り出している．

当時，日本では，前年の1976年に出たNECのTK-80の売れ行きのよいのに触発された大手企業から，続々とマイコンキットが発売されている．ところが，SORD社のM200シリーズは，PETと似た形状で8インチFDD搭載の実用パソコンだったのである．

SORD社が本当に成功しはじめたのは，1980年2月に発表した「SORD-PIPS」という簡易言語が有名になってからである．PIPSは，事務処理用の

簡易言語であり，当時使われていたBASICにくらべると，ソフトウェアの生産効率が大幅に向上したのである．PIPSの開発者は，日銀の望月宏氏でありBASICで書かれていた．これをSORD社が買いとってアセンブラにし，逆スクロールなどの機能を加えて商品化したのがタン氏である．同氏は，ベトナム出身であることで，話題を提供したものである．

実際，このころのSORD社は非常に国際色豊かで，数人のベトナム出身者以外に，台湾，ニュージーランド，アイルランド，イギリスなどの人がいた．新卒者の採用も1981年には30人，1982年には80人，1983・84年には100人と増えていったが，人気があったため応募者も多く，入社は10倍以上の非常に狭き門となっていた．

ハードとしては，1978年から販売しているM223があるが，これは米国におけるアルテア8800以来のS-100バスとRS-232Cを標準装備していることを売りものにし，5インチFDDを2台，CPUにはZ80Aをのせていた．応用分野としては，アナログ・デジタル入力インターフェースボードを備え，データ処理用のビジネスマシンをねらっていた．

これに対し，1980年に発表したM243（153万円）は，マルチタスク，漢字処理，通信機能，PIPSをのせて，「ビジネスマシン」という新しいコンセプトを提唱した．OSについても，異種言語を同時に走らせるということに加え，マルチタスクを可能にするなど，非常に早くから理想形をねらっていた．M243のCPUはZ80であり，言語としてはBASICをはじめ，リロケータブルなアセンブラ，FORTRAN IV，PASCALを積んでいた．M243は非常によいマシンであったが，惜しいことに価格が高く，アマチュアが家庭でたのしむためには使えなかった．

米国では，BASICがはやったあと，デジタル・リサーチ社の作った8080およびZ80用のCP/MというOSが広く使われた．日本の8ビット機では，富士通のFM 8にオプションでついていた程度で，CP/Mはほとんど使われず，いっきに1981年からはじまる16ビット機によるMS-DOSの時代に移行する．

SORD社は，16ビット機では，1982年にインテル社の8086を採用し，RM-DOSをのせたM343を出し，1983年にはモトローラ社の68000とザイログ社のZ80Aを使ったデュアルCPU機，M68を出している．M68には，CP/M 68Kを採用した．さらにこの年，32ビット機のM685を発表している．

1982年には，任天堂のファミコンなどのゲーム機の元祖ともいえるM5（4万9800円）を売り出している．これは，ROMカセットを上から差し込むようになっている．コネクタは，28ピンの2列である．設計思想は，コモドールと似ており，VIDEO，SOUNDの出力以外に，家庭用のテレビにつなげるようRF端子もついている．また，左右のゲーム用コントローラ（ジョイスティック）の端子がある．外部メモリはカセットレコーダで，プリンタ用端子もついている．電源は，やはり外付けで，丸型の6ピンを使っている．当然ながらキーボードもついていて，しかも4段にかわる（大文字，小文字，グラフィック，命令など）といった全体的に凝ったものであった．

1985年になると，のちに東芝のJ-3100に大きな影響を与えたIS-11Cも出している．全体的にみてSORD社は，世の中のニーズよりはやく製品を出しすぎたともいえるし，新しいマーケットを開発し続けたともいえよう．SORD社は現在，東芝の系列に入っており，検見川にショールームをもっている．

椎名氏はその後，SORD社を辞め，新しくプロサイド社を設立し，PC-9801とIBM PC/ATの両互換機や，80486を使ったボードなど，新しい世の中の動きに向けた新製品の開発にチャレンジしている．

●SORD社の新技術採用

1974年	SORD BASIC
1975年	RPG2
1976年	日本文章発生装置
1977年	8インチFDD，S-100バスの採用（M200）
1978年	マルチユーザーOS　簡易言語（PIPS）の採用（M200）
1980年	RM-DOS（RT/MT/MU）（8086用）
1982年	ノートブック型の元祖（M5）
1983年	内部32ビットIC（68000）の採用（M68）

●SORD社の主要製品系列

1972年	SMP 80/08（8008）
1974年	SMP 80/20（8080）
1976年	SMP 80/50（Z80）
1977年	M200
1978年	M223，M203，M100
1979年	M100ACE（FDD付）
1980年	M243（SORD-PIPS）
1981年	M23，M416（8086）
1982年	M23P，M343，M5
1983年	M685（32ビット），M68，M243EX
1984年	M343SX
1985年	M68MX，IS-11C，UNIBOX（1985年2月より東芝系列に入る）
1986年	UNIBOX2
1987年	FUTURE32
1988年	SR-3300，MS40
1989年	FUTURE32，HR-M12/M15/M12F

祐安重夫

プログラマの道具学

第6回　ねずみ

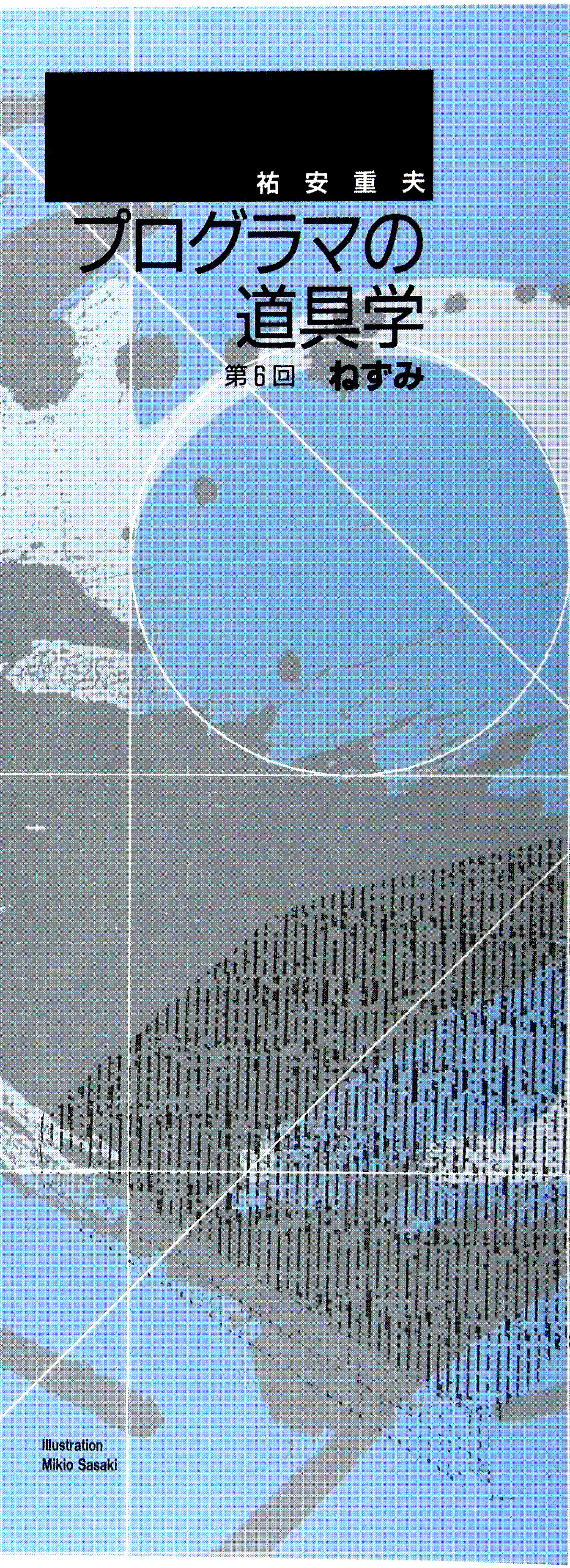


Illustration
Mikio Sasaki

机の上に3匹のねずみ――マウスがいて，それぞれMacintosh，PC-9801，NEWSの3つのコンピュータに接続されている．マウスには押ボタンがついていて，Macintoshのものは1つ，PC-9801は2つ，NEWSは3つとなっている．さらに3匹のねずみは，形もすこしずつちがっているので，すぐに区別できるはずなのだが，ときどきMacintoshをNEWSのマウスで操作しようとしてしまうなどということがある．

コンピュータのそばで最初にねずみの飼育をおこなったのは，アメリカのSRI(Stanford Research Institute)だったそうだが，それは手のひらよりもかなりおおきく，また机の上をなめらかにすべらなかったためか，普及しなかった．

マウスに注目して，そのブームのきっかけをつくったのは，XeroxのPalo Alto研究所でSmalltalk言語を開発し，それをAltoというパーソナル・コンピュータにのせた，Alan Kayという人物である．Altoは実用機として市場にでることはなかったが，その後XeroxはStarというオフィス用のワークステーションを開発し，そこでもマウスはつかわれた．またSmalltalkそのものも，現在ではさまざまなコンピュータの上で動作するようになっている．

AltoやSmalltalkによって提示されたのは，オブジェクト指向という概念であり，それにはマウスは格好の道具だった．画面上の対象を指示してボタンをおすという操作は，オブジェクトに対してメッセージをおくることのアナロジーとみなされたのである．

AltoやStarのコンセプトにおおきな影響をうけてうまれたのが，AppleのLisaとよばれるパーソナル・コンピュータである．Lisaそのものはあまりうれず，結局失敗におわったが，その後継機であるMacintoshは世界的



なベスト・セラーとなり，おおくのユーザを獲得した．

Macintoshはそれまでは高価なワークステーションなどでしか利用できなかった，マウスをつかった高度なユーザ・インターフェースを，個人の手のとどく場所にもってきた．

その結果として，現在ではほとんどのパーソナル・コンピュータに，ねずみを飼育するための環境が用意されるようになっている．

ワークステーションではX-Windowに代表されるような，マウスを活用したマルチ・ウィンドウ・システムなどのソフトウェアがくみこまれていて，その上でさまざまな機能を実行することができる．またMacintoshは，最初からシステムをマウスなしでは動作しないように構成しており，快適で良好なユーザ・インターフェースが実現されている．極端な場合には，キーボードなしでマウスだけでコンピュータを操作することさえ，Macintoshでは可能なことがある．

残念なことにMS-DOSを利用したPC-9801などのパーソナル・コンピュータでは，システムそのものにはマウスを利用するための機構はくみこまれていない．現状ではアプリケーション・プログラムが個別に対応しているだけであり，マウスがつかえるソフトウェアとつかえないソフトウェアがある．

またMacintoshではあらかじめシステムがマウスの標準的な機能を規定しているが，MS-DOS上のアプリケーションでは，ソフトウェアを開発する人間が勝手にきめているので，それぞれのプログラムでねずみがまったくちがう行動をおこして，ユーザを混乱させることもある．

もっともMS-DOSでも，マルチ・ウィンドウ環境であるWindowsが，普及のきざしをみせており，ここではマウスも本格的に利用できるようである．

ワークステーションの場合は，たいていはUnixというねずみの生息を前提としていないシステムを利用しているが，Unixはもともとかなりの柔軟性をもっていたこともあって，それなりにねずみとなかよくやっているようだ．

プログラム遊工房

谷口典生

六乃巻
「タイピング練習」ゲーム

今回は，タイピング練習用のゲームを作ってみました．文章，ワード単位での練習プログラムではないので本格的な訓練には使えませんが，ゲーム感覚で楽しんでもらえればよいと思います．ワンポイントテクニックでは，プログラムを読みやすくするために記号定数を使うことについて書いています．

リスト1

```
/*
 * typegame.c          タイピング練習ゲーム
 */

#include <time.h>
#include <conio.h>
#include <stdlib.h>

#define LANES           8                  /* レーン数 */
#define G_SLEEP         0                  /* 泥棒がいない */
#define G_FORWARD       1                  /* 泥棒が下に向かっている */
#define G_BACKWARD      2                  /* 泥棒が上に向かっている */
#define CHR_CLR         ' '
#define CHR_MONEY       '$'                /* お金をあらわすキャラクタ */
#define gotoxy(x,y)     printf("¥x1b[%d;%dH",y,x)

const char chars[] = "abcdefghijklmnopqrstuvwxyz,.";    /* 練習する文字 */

struct lane_struct {
        int x;                             /* レーンのX座標 */
        int gang_y;                        /* 泥棒のY座標 */
        int gang_stat;                     /* 泥棒の状態 */
        int gang_chr;                      /* 泥棒の文字 */
        int gang_speed;                    /* 泥棒の速度 */
        int gang_spcnt;                    /* カウンタ */
        int money_y;                       /* お金のY座標 */
} lane[LANES];

int score;                         /* 得点 */
int waitval;                       /* 時間待ち用の値(コマンドライン引数) */

void putxy(int x, int y, int c)            /* 座標 (X,Y) に文字を表示する */
{
        gotoxy(x,y);
        putch(c);
}

void print_score()                         /* 得点表示 */
{
        gotoxy(3,7);
        printf("SCORE¥n %5d", score);
}

void game_init()                           /* 初期化ルーチン */
{
        int i;

        for(i=0; i<LANES; i++) {
                lane[i].x = i*5+15;                /* 各レーンのX座標 */
                lane[i].gang_stat = G_SLEEP;
                lane[i].money_y = 20;              /* お金の初期位置 */
                putxy(lane[i].x, lane[i].money_y, CHR_MONEY);
        }
        score = 0;
        print_score();
}

/* 指定したレーンの泥棒を動かす */
void move_gang(struct lane_struct *lane, int hitkey)
{
        if(lane->gang_stat==G_SLEEP) {  /* 泥棒がまだ動いていない */
                if((rand()%20)==1) {    /* 20分の1の確率で泥棒を出す */
                        lane->gang_stat = G_FORWARD;
                        lane->gang_y = 1;
                        lane->gang_chr = chars[rand()%(strlen(chars))];
                        lane->gang_speed = 10 - (rand()%6);
                        lane->gang_spcnt = 0;
                        putxy(lane->x, lane->gang_y, lane->gang_chr);
                }
                return;
        }

        /* 入力したキーが活動中の泥棒に当ったか? */
        if(lane->gang_chr==hitkey) {
                lane->gang_stat = G_SLEEP;      /* 泥棒を消す */
                putxy(lane->x, lane->gang_y, CHR_CLR);
                score++;                        /* 得点加算 */
                print_score();
                return;
        }

        /* 各泥棒の速度によってカウンタを加算し、動かすかどうかを決める */
        lane->gang_spcnt++;
        if(lane->gang_spcnt < lane->gang_speed)
                return;
        lane->gang_spcnt = 0;

        if(lane->gang_stat==G_FORWARD) {
                /* 下へ移動 */
                putxy(lane->x, lane->gang_y, CHR_CLR);
```

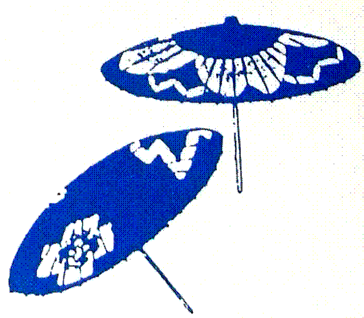
プログラムの使い方

このプログラムはPC-9800シリーズ用で，Turbo C，QuickCおよびMS-Cに対応しています．リスト1を打ち込み，コンパイルして実行ファイルを作ってください．

起動するときは，

TYPEGAME {wait_time}

と，待ち時間を指定します．この値は小さければ早く，大きければ遅くなりますので，使用機種の速度によって値を調整してください．V30-8MHzで50，80286-10MHzで100，80386-20MHzで200くらいの値を目安にして，好みに合わせて増減してみてください．



```
                lane->gang_y++;
                putxy(lane->x, lane->gang_y, lane->gang_chr);
                if(lane->gang_y==lane->money_y-1)       /* 金に接触したか? */
                        lane->gang_stat = G_BACKWARD;
        } else {                        /* lane->gang_stat==G_BACKWARD */
                /* (金を引きずって)上へ移動 */
                putxy(lane->x, lane->money_y, CHR_CLR);
                lane->gang_y--;
                lane->money_y--;
                putxy(lane->x, lane->gang_y, lane->gang_chr);
                putxy(lane->x, lane->money_y, CHR_MONEY);
        }
}

void wait(int val)                      /* 時間待ちルーチン */
{
        int i,j;

        for(i=0; i<waitval; i++)
                for(j=0; j<val; j++);           /* 空ループ */
}

void game()                     /* ゲームのメインルーチン */
{
        int i, key;

        game_init();                            /* 変数の初期化 */

        while(1) {
                if(kbhit())
                        key = getch();
                else
                        key = -1;

                for(i=0; i<LANES; i++) {        /* レーンの数だけループ */
                        move_gang(&lane[i], key);
                        if(lane[i].money_y<=2)
                                return;         /* 金が上まで持っていかれた */
                }

                wait(10000/(score+100));        /* 得点が増すと速度が上がる */
        }
}

int main(int ac, char *av[])
{
        time_t seedtime;
        int key;
        int hiscore=0;                          /* 最高得点 */

        if(ac!=2) {
                printf("usage : typegame {wait_time}¥n");
                exit(0);
        }
        waitval = atoi(av[1]);                  /* 引数(レベル数)を取得 */

        time(&seedtime);
        srand((unsigned int)seedtime);          /* ランダマイズ処理 */

        do {
                printf("¥x1b*¥x1b[>5h");        /* 画面クリア、カーソル消去 */
                gotoxy(1,4);
                printf("HI-SCORE¥n %5d", hiscore);
                game();                         /* ゲーム実行 */
                if(score>hiscore)
                        hiscore = score;
                gotoxy(1,22);
                printf("GAME OVER¥ntry again?(Y/N)");
                while(kbhit())
                        getch();
                do {
                        key = getch();
                } while(toupper(key)!='Y' && toupper(key)!='N');
        } while(toupper(key)=='Y');

        printf("¥x1b[>5l");                     /* カーソル復帰 */
}
```

ゲームの遊び方

プログラムを実行すると，画面上方からパラパラと泥棒（画面上ではアルファベット文字）が降りてきます．下にはお金（画面上では「$」）が置いてあり，上から降りてきた泥棒が下まで到達すると，こんどはお金をひきずって上へと戻っていきます．お金がどれか１つでもいちばん上までもっていかれてしまうとゲームオーバーです．プレイヤーはそれを阻止するために，降りてくる泥棒と同じ文字をキーボードから入力して撃退します．入力されたキーと同じ文字は画面上から消え，得点になります．ただしお金を上へひきずっている途中の泥棒を消した場合，お金はそこに置き去りにされます（下には戻らない）．お金が高い位置にあれば，当然その上から降りてくる泥棒は早くお金に到達することになるので，とくに注意しないといけません．また，泥棒の速度はまちまちなので，単純にでてきた順番に撃っていると，足の速い泥棒にお金をかすめ取られてしまうこともあります．

得点が上がるにつれて，徐々に泥棒の速度は上がっていきます．だいたい100点で最初の２倍，200点で最初の３倍くらいのスピードになりますから，起動時に指定する引数は少し大きめにして，最初はゆっくり目にはじまるようにしておくとよいでしょう．

ゲームオーバーになると，もう１度やるかどうか聞いてくるので，もう１度プレイするときには「Y」を，終了するときに

は「N」を入力してください．続けてプレイすると，それまでのハイスコアが更新されて表示されます．

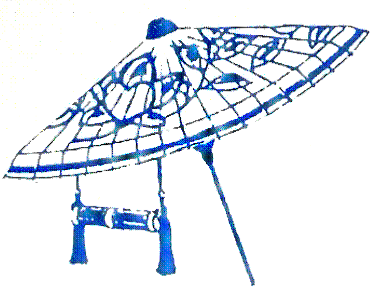
練習用文字のカスタマイズ

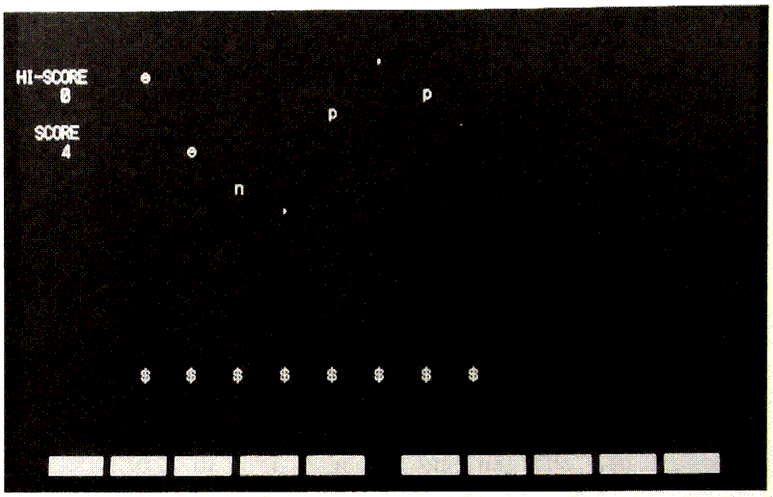
リスト１では，アルファベットの小文字（「a」～「z」）とカンマ（「,」），ピリオド（「.」）の28文字が練習できるようになっています．

これはリスト１の17行目で設定しているので，ここを変更することによって，練習する文字を自由に選ぶことができます．

たとえば，アルファベットの大文字も同時に練習したければリスト２の上のように，数字を入れたければリスト２の下のように変更します．

同様にしてカナや記号も練習することができます．文字列定数の中ではダブルクォーテーション（「"」）や円記号（「¥」）はそれぞれ「¥"」，「¥¥」と書くことに注意しましょう．





リスト2

```
const char chars[] = "abcdefghijklmnopqrstuvwxyz,.ABCDEFGHIJKLMNOPQRSTUVWXYZ";
const char chars[] = "abcdefghijklmnopqrstuvwxyz,.0123456789";
```

プログラムの説明

9-14行

各種定数の定義．

9行

レーン数（＝出現する敵の数）を定義します．スピードを遅めにしても難しく感じる場合は，この値を減らしてみてください．もちろん，この値を増やせば敵の数が増えてゲームは難しくなります．

10-12行

泥棒の状態定数を定義します．

13-14行

空白文字とお金を表示するための文字を定義します．

15行

表示位置をy行xカラムに移動させる関数gotoxy（x, y）を定義します．

17行

タイピングの練習に使う文字を文字列定数chars[]に設定します．

19-27行

各レーンの情報を格納する構造体を宣言します．

putxy()

座標（x，y）に文字cを表示する関数です．

print_score()

現在の得点を画面左に表示します．

game_init()

各レーンのデータを初期化し，お金を画面に表示します．また，得点をクリアして表示します．

move_gang()

指定したレーンの泥棒を動かします．

61-71行

泥棒が動いていない（G_SLEEP）状態のときは，20分の１の確率で新たに泥棒を発生させます．泥棒の位置を最上端に設定し，泥棒のキャラクタをchars

[]の中からランダムに選びます．泥棒のスピードも5～10の間でランダムに設定します．

74-80行

泥棒が活動中（G_FORWARDまたはG_BACKWARD）のときは，泥棒のキャラクタとキー入力された文字をくらべ，一致した場合には泥棒を消し，得点を加算します．

83-86行

gang_speedは，move_gang（）が何回呼ばれるごとに泥棒を実際に動かすかで速度をあらわしています．ここでは，カウンタgang_spcntを使ってこれを実現しています．

88-94行

泥棒が下に移動中（G_FORWARD）であれば，泥棒のキャラクタを1つ下に動かし，もしもそのレーンの金の手前の位置にきたら泥棒の状態をG_BACKWARDにします．

95-102行

泥棒が上に移動中（G_BACKWARD）であれば，泥棒とそのすぐ下にあるお金のキャラクタを1つ上に移動します．

❖wait()

時間待ちルーチンです．引数とwaitval（コマンドラインで指定した待ち時間の値）によって空ループを回しています．

❖game()

119-130行

ゲームのメインループです．

120-123行

キー入力があれば入力された文字をkeyに代入します．キー入力がなければ－1を代入して，いずれの文字とも一致しないようにします．

125-129行

レーン数だけループし，move_gang（）を呼びます．お金が上までもち去られたらゲームオーバーなのでgame（）の呼び出し先に戻ります．

131行

時間待ちルーチンを呼びます．このときの引数は，得点が増えるにしたがって小さくなるようにして，高得点になるとゲームの速度が上がるようにしています．

❖main()

139行

ハイスコアを0に初期化しておきます．

141-144行

コマンドラインで引数が指定されなかった場合は，かんたんな起動方法を表示してプログラムを終了します．

145行

引数をint型に変換してwaitvalに代入します．

147-148行

time関数を利用して乱数の初期化を行います．

151-153行

画面クリア，カーソルを消去し，ハイスコアを画面左上に表示します．

154行

ゲーム実行ルーチンを呼び出します．

155-156行

ハイスコアの更新処理を行います．

157-160行

ゲームオーバーのメッセージを表示し，キー入力バッファをクリアします．

161-163行

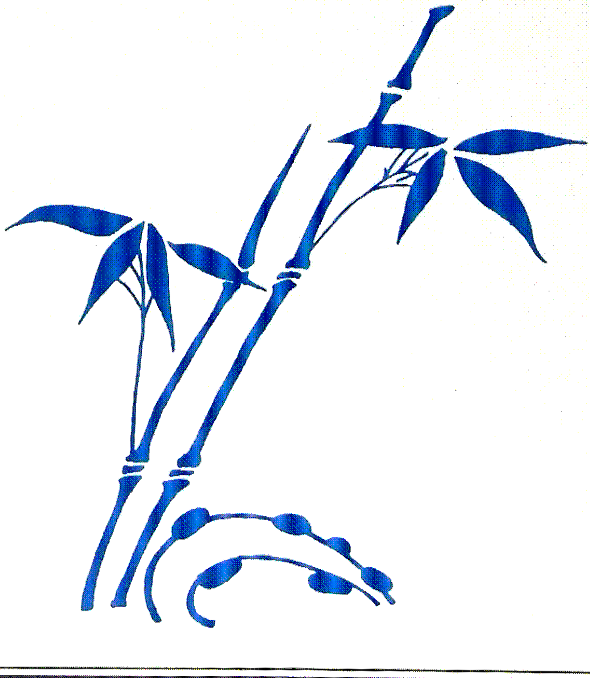
YかNのキーが入力されるまでループします．

164行

Yが入力されたらリプレイします．

166行

カーソルを表示し，プログラムを終了します．



ワンポイントテクニック

記号定数を利用してプログラムを読みやすくしよう

リスト1の10～12行目を見てください．G_SLEEP, G_FORWARD, G_BACKWARDの3つの定数には，構造体lane_struct中のgang_statが泥棒の状態をあらわすために取り得る値が定義されています．gang_statの取る値は，3つの状態を区別できればよいのですから，わざわざG_SLEEPなどの名前に定義せず，そのまま0などの値をプログラムに書いてもよいし，値そのものも0，1，2である必要はありません．

しかし，プログラムをあとで読み直したり変更したりすることを考えると，値に意味をもたない数字がそのままプログラムの中にあるのは好ましくありません．前述のような変数を，値を代入しているところからはなれたところで参照している場合を考えれば，容易に想像がつくと思います．

詳しくコメントをつけることが，少しは理解のたしになるかもしれませんが，まずプログラムを読みやすく書くことが先決です．このような変数がとる値は，いってみればラベルのようなものですから，できるだけ意味のある名前をもった記号定数に置き換えるように心がけましょう．

記号定数の名前のつけ方は変数の名前のつけ方と同様，人それぞれで違うと思います．英語でもローマ字でもかまいませんが，自分なりに一貫したルールを作り，それにしたがって名前をつけていけば，あとで混乱せずにすみます．同じグループの定数名には，同じヘッダ（前述の例では「G_」）をつけるとわかりやすくなると思います．

前述の例では定数の数は3つですが，もっと数が多いとき，たとえば曜日や月の名前を記号定数にしたいときなどは，#define文がずらっと並ぶことになります（図1）．このように名前から指す内容が明らかにわかり，コメントをつける必要もないようなときは図2のように，列挙型を使って1行に書いてしまうのも有用な方法です．列挙型で書き並べた名前はint型の定数として順に0，1，2，……の値を取ります（明示的に値を指定しなかった場合）から，実用上は図1のように定義された定数と考えてよいのです．

列挙型の変数というのも，宣言して使用することができます（図3）．しかし，列挙型の変数は実際には単なるint型で，コンパイラはこのように宣言されたweekに代入される値はチェックしないので注意が必要です．

記号定数を使うべき場面はもっとたくさんあります．リスト1について見ても，たとえば，得点表示のXY座標（40行目，152行目など），各レーンのX座標の初期値と間隔（49行目），お金の初期Y座標（51行目）などに使っている数値も，本来は#define文を使って記号定数として定義しておいたほうがよいと思います．ただ，ここで紹介するプログラムは200行前後の短いもので，あまり#define文を多用するとかえってわずらわしくなるため，変更する可能性のある定数や複数か所で使われる定数などに限って記号定数に置き換えるようにしています．

大きめのプログラムを書くときには，記号定数の定義などはまとめて別のファイルにし，#include文で取り込むなどとすると管理しやすくなります．

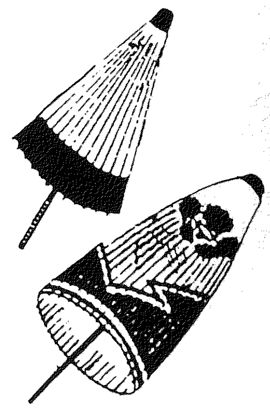


図1　曜日をあらわす定数

```
#define W_MON   0
#define W_TUE   1
#define W_WED   2
#define W_THU   3
#define W_FRI   4
#define W_SAT   5
#define W_SUN   6
```

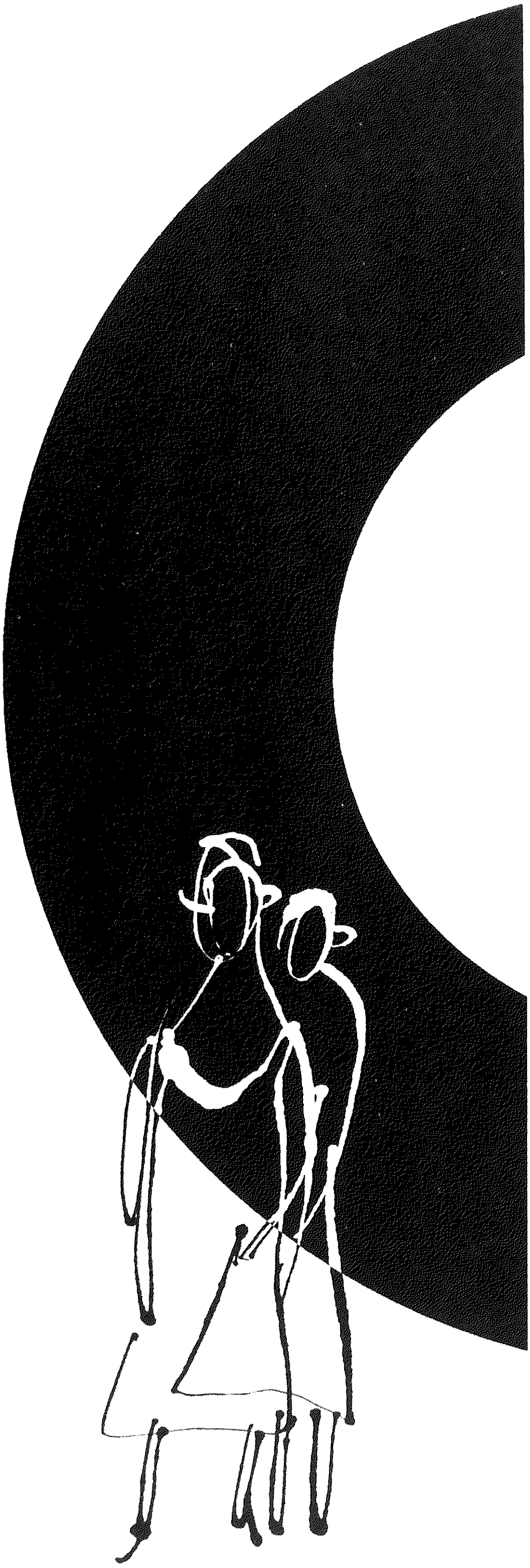
図2　列挙型定数による表現

```
enum weekday { W_MON, W_TUE, W_WED, W_THU, W_FRI, W_SAT, W_SUN };
```

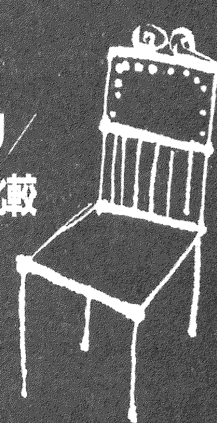
図3　列挙型の変数も宣言できるが‥‥

```
enum weekday week;
```

Illustration : Masami Nozue



実践C塾 第6回



システムコールライブラリ／日本語処理ライブラリの比較

塾頭 水居　浄

前回にひき続いて，Microsoft CとTurbo Cのランタイムライブラリにおける，処理系固有のライブラリ関数の比較検討を行います．今回は，システムコールライブラリの残りと，日本語処理ライブラリについて検討します．
とくに，MS-DOSのCTRL+Cによるプログラム終了機能や致命的エラーをアプリケーションプログラム内で処理する際の問題点について解説します．

ワンラインNEWS
電子機器およびコンピュータソフトメーカーのサン電子はIBM PC/AT互換のブック型パソコンを独自開発し，OEM供給を開始した．

CTRL+C割り込みの処置

CTRL+C機能を無効にする

MS-DOSでは，一般に［CTRL］+［C］のキーを押すことによって実行中のプログラムを終了させる機能があります．これは，MS-DOSのシステムコールの実行中に［CTRL］+［C］が押されると，INT 23Hベクタの割り込みハンドラに制御が渡されるという機構によって実現されています．INT 23Hベクタの割り込みハンドラは，標準では実行中のアプリケーションを終了させるように設定されているわけです．

もちろん，アプリケーションによっては，［CTRL］+［C］で終了してしまっては困るというケースも多くあります．たとえば，一般的なスクリーンエディタでは［CTRL］+［C］をエディタの制御コマンドに使っています．こういう場合は，［CTRL］+［C］で終了してしまわないようにするための処置を行わなくてはなりません．

また，［CTRL］+［C］でプログラムが終了するのはよいとしても，その際にオープンしていたファイルなどのあと始末をしてから終了させたい，という場合もあります．［CTRL］+［C］キーが押されたときに，こういった通常とは異なる処理を実行したい場合は，プログラムはINT 23Hベクタをフックして，自前のハンドラを用意する必要があります．

INT 23Hベクタをフックする場合，Microsoft Cでは_dos_setvect，Turbo Cではsetvectといった関数を使って，割り込みベクタを書き換えてしまうこともできますが，もっとかんたんで移植性のよい方法が両処理系には用意されています．それは，ANSI規格の関数でもあるsignal関数です．

リスト1のプログラムは，signal関数のかんたんな使用例です．このプログラムは，標準入力から入力された文字をそのまま標準出力に出力する，いわゆるcatプログラムのごくかんたんなものです．catプログラムは一般には，［CTRL］+［C］が押されると処理を終了しますが，このプログラムではsignal関数を使ってハンドラを設定しているため，［CTRL］+［C］を押しても処理を終了せず，文字列を表示して実行を継続します．このプログラムでは，ハンドラ内でもう一度signal関数を呼び出してハンドラを設定しなおしていますが，これは，［CTRL］+［C］が押されると，MS-DOSはハンドラを呼び出す直前に，ベクタのアドレスをもともとのハンドラのアドレスに戻してしまうからです．ハンドラ内で設定しなおさないと，2回め以降の［CTRL］+［C］ではプログラムが終了するようになってしまいます．

単に［CTRL］+［C］による中断を抑止したいだけの場合は，わざわざハンドラ関数を作成しなくても，signalの2番目の引数に特別な定数SIG_IGNを指定することができます．これを指定すると，［CTRL］+［C］が押されてもプログラムが終了しなくなります．また，もとの状態に戻したい場合は，SIG_DFLを指定することができます．

このsignalの機構を利用すると，たとえば，テンポラリファイルを作成して使用するようなプログラムで，［CTRL］+［C］が押されたときに，テンポラリファイルをすべて削除してから処理を終了するといったことができるわけです．実際，Microsoft Cコンパイラ自体も，そういう風にして［CTRL］+［C］が押されたときにテンポラリファイルを削除しています．

signalはANSI規格の関数ですから移植性がよいのですが，実際には［CTRL］+［C］以外の割り込み，たとえば数値演算エラー割り込みなどの処理機構も組み込まれていますので，単に，［CTRL］+［C］を無視したいといっただけの用途には，少々大がかりすぎるきらいもあります．こういった点を配慮したのだと思われますが，Turbo Cはctrlbrkという独自の関数でも，［CTRL］+［C］の処理を制御することができるようになっています．

なお，スクリーンエディタなどのように，［CTRL］+［C］をほかのキー入力と同等に扱いたいという場合は，signalで［CTRL］+［C］割り込みを無視するように設定するだけでは十分

リスト1　signalを使った［CTRL］+［C］の処理の例

```
#include <signal.h>
#include <stdio.h>

void handler()
{
    printf("CTRL+C pressed. ¥n");
    signal(SIGINT, handler);
}

main()
{
    int c;

    signal(SIGINT, handler);
    while ((c = getchar()) != EOF)
        putchar(c);
}
```

ではありません．この場合，INT 23Hをフックして処理が中断しないようにしているだけなのですから，[CTRL]+[C]を特別なキーストロークとして，MS-DOSが処理してしまう点には変わりがないわけです．こういう場合は，MS-DOSのINT 21Hファンクション06Hを使ってキー入力を処理するのがもっともかんたんな解決方法です．このファンクションは，[CTRL]+［C］を特別扱いしないので，ほかのキーと同様に読みとることが可能になります．

リスト2に，このファンクション06Hを使ったキー入力関数の例を示します．このファンクションはゼロフラグによって状態を返すので，bdos関数やintdos関数などでは正確な入力の判定ができませんから，インラインアセンブラを使用しています．Microsoft CとTurbo Cではキーワードが異なるので，条件コンパイルで対処しています(__TURBOC__はTurbo Cのシステム定義のマクロです)．また，V.6.0より前のMicrosoft Cではインラインアセンブラが使用できないので，この関数はMASMで作成する必要があります．

リスト2　MS-DOSファンクション06Hを使ったキー入力関数

```
#ifdef __TURBOC__
  /* Turbo C の場合は _asm を asm に変更 */
  #define _asm asm
#endif

int getkey(void)
{
     _asm mov     AH,06h
     _asm mov     DL,0FFh
     _asm int     21h
     _asm jz      NONE
     _asm sub     AH,AH
     _asm jmp     short END
NONE:
     _asm mov     AX,-1
END:
     ; /* return AX
       *     0～0xFF 入力された文字
       *     -1      入力なし
       */
}
```

なお，キー入力の処理をファンクション06Hで行っていても，画面出力などをMS-DOSのシステムコールで行っていると，やはり［CTRL]+［C］によってプログラムが中断してしまうことがあります．したがって，この場合もsignalなどでINT 23Hベクタをフックしておく必要性が残るという点に注意してください．

致命的エラーの処置
自前のハンドラで処理する

致命的エラーというのは，MS-DOSでディスクの入っていないフロッピーディスクドライブにアクセスしたときなどに発生するものです．通常は次のようなメッセージが表示されて，ユーザーに処置を尋ねてきます．

ドライブの準備ができていません．<読取り中><ドライブ A:>
中止 <A>，もう一度 <R>，無視 <I> ?

このような致命的エラーが発生したとき，MS-DOSはINT 24Hベクタの割り込みハンドラに制御を渡します．通常，この割り込みハンドラは，COMMAND.COM中にあり，上記のメッセージを表示して処置をユーザーに尋ねるというプログラムになっています．したがって，通常はこの致命的エラーのハンドリングはデフォルトのままにしておいて，COMMAND.COM中のハンドラに処理を任せておけば十分なのですが，たとえばスクリーンエディタのように画面を制御しているプログラムなどでは，このようなメッセージが表示されるのはあまり歓迎できないという場合もあります．また，ユーザーに処置を尋ねずにそのままエラーを返してくれたほうがありがたいという場合もあるでしょう．そういった要求を満たすためには，INT 24Hベクタをフックして，致命的エラーを自前のハンドラで処理するようにしなくてはなりません．

INT 24Hベクタのフックも，_dos_setvectやsetvectで行えますが，よりかんたんに処理できるように，両処理系とも専用の関数が用意されています(表1)．

すなわち，_harderr関数を使って割り込みハンドラを設定し，割り込みハンドラ中では_hardresumeを使ってMS-DOSに復帰するか，_hardretnを使ってアプリケーションへ直接復帰します(Turbo Cの場合はそれぞれharderr，hardresume，hardretn関数になります)．リスト3に致命的エラーハンドラの例を示します．

Microsoft CとTurbo Cの両者では，関数名以外に，ハンドラ関数やその引数の型が異なります．Microsoft Cのハンドラはつねにfar関数ですが，Turbo Cのハンドラはメモリモデルのデフォルトです．引数は順序が異なるだけで，ほとんど同じものです．したがって，リスト3のように条件コンパイルを記述すれば，両処理系でソー

スを共通化することが可能です．

ハンドラ関数内では，MS-DOSのINT 21Hシステムコールは，0CH以下の番号のファンクションしか使用できないという制限があります．したがって，ハンドラ関数内ではprintfなどのstdio.h系の関数は使用できません．cprintfなどのconio.h系の関数なら使用できます．このプログラムのハンドラ関数では，エラーコードを表示したあと，ユーザーに処置の方法を尋ねてきます．ここでAのキーを押すとプログラムはただちに終了します．Rを押すと，MS-DOSがリトライを行います．Fを押すと，この致命的エラーを発生したシステムコールが失敗したものとして，MS-DOSを経由せずにプログラムに復帰します．

COMMAND.COM内のハンドラでは「無視<I>」(Ignore) という選択肢がありますが，このハンドラではそれは選択できないようにして，かわりに「失敗」(Fail) を選択できるようにしている点に注意してください．Ignoreという選択は，MS-DOSがそのシステムコールが成功したものとみなして次の処理を行うというもので，一般にはこれを選択することは，正常な動作結果が保証されない危険な操作になるのです．COMMAND.COM内のハンドラがなぜこのような選択肢を表示するのかは，歴史的な経緯によるもので，あまり必然性はありません．実際，MS-DOS 3.3 (IBM PC版のV.3.3のこと．PC-9801のV.3.3の中身はV.3.2相当品) 以降のCOMMAND.COMでは，Ignoreは選択肢から外され，かわりにFailが選択できるようになっています．

_hardresumeと_hardretnの使い分けですが，一般にAbort，Retry，Ignoreを選択する場合には_hardresumeを使用します．_hardretnはFailを選択する場合に使ってください．実は，MS-DOS 3.1以降では_hardresumeの引数に_HARDERR_FAIL (== 3) を指定することも可能なのですが，旧バージョンのMS-DOS上で正常に動作しなくなります．

BIOSライブラリ
機種依存したBIOSを呼び出す

現実のアプリケーションプログラムでは，機種依存したBIOSを直接呼び出すことが必要なケースも多くあります．そこで，両処理系ともBIOSとのインターフェースをサポートしたライブラリを提供しています．表2に，両処

表1　INT 24Hベクタのフック専用の関数

Microsoft C	Turbo C	機能
_harderr	harderr	INT 24H割り込みハンドラの設定
_hardresume	hardresume	ハンドラからMS-DOSへ復帰
_hardretn	hardretn	ハンドラから直接アプリケーションへ復帰

リスト3　致命的エラーハンドラの例

```
#include <conio.h>
#include <dos.h>
#ifdef __TURBOC__
  #define _harderr       harderr
  #define _hardresume    hardresume
  #define _hardretn      hardretn
  #define _HARDERR_ABORT 2
  #define _HARDERR_RETRY 1
  int handler(int doserr, int deverr, int bp, int si)
#else
  void far handler(unsigned deverr, unsigned doserr, unsigned far *devhdr)
#endif
{
    int     ch;

    cprintf("¥r¥nデバイスエラーコード: %d", deverr);
    cprintf("¥r¥nMS-DOS  エラーコード: %d", doserr);
    while (1) {
        cprintf("¥r¥n(A)bort, (R)etry, or (F)ail? ");
        ch = getch();
        switch (ch) {
        case 'A':
        case 'a':         /* Abort program */
            _hardresume(_HARDERR_ABORT);
            break;
        case 'R':
        case 'r':         /* Try again */
            _hardresume(_HARDERR_RETRY);
            break;
        case 'F':
        case 'f':         /* Return to program with error code */
            _hardretn(doserr);
            break;
        }
    }
}


main()
{
    /* ハンドラの登録 */
    _harderr(handler);

    /* テスト */
    cprintf("ドライブ B: にフロッピィが入っていない状態にして¥r¥n" );
    cprintf("どれかキーを押してください．" );
    getch();
    if (mkdir("B:¥¥test") != 0) {
        cprintf("¥r¥nドライブ B: のアクセスに失敗しました．¥r¥n");
        exit(1);
    } else {
        cprintf("¥r¥nドライブ B: のアクセスに成功しました．¥r¥n");
        rmdir("B:¥¥test");
        exit(0);
    }
}
```

理系のPC-9801版のBIOSライブラリ関数を示します．

Microsoft C 5.1では，この種のライブラリは提供していませんでした．Microsoft C 6.0のBIOSライブラリも，どちらかというとIBM PC版のソフトの移植しやすさを考慮しただけといった印象で，PC-9801に適応した使いやすいものになっているとはいえないようです．

一方，Turbo Cでは2.0からTurbo C++へのバージョンアップの際に，全体的な仕様の見直しがなされています．全体的に，よりきめ細かく使いやすくなるように配慮されているといえます．もちろん，Turbo C++ではTurbo C 2.0互換のライブラリも提供されていますので，上位互換性の問題もクリアしています．

この種のライブラリは，従来はユーザーが一から作らなくてはならなかったのですが，処理系が用意してくれていることでかなり手間が省けるようになってきました．しかし，これらのライブラリはすべて処理系に依存してしまいますから，移植性を考慮する場合は注意が必要です．

日本語処理ライブラリ
マイクロソフト社が基本的な仕様を作成

Cコンパイラを使って日本語MS-DOS上で動作するアプリケーションを作成する場合に，大きな問題となったのが日本語処理の方法です．日本語の文字コード自体は「MS漢字コード」（いわゆるシフトJISコード）を採用したため，文字コードとして1バイトコードと2バイトコードが混在する結果となり，また，とくに全角文字の2バイト目のコードが，アルファベットなどの1バイトコードと一致してしまうケースがあり，単純には処理できなくなってしまったのです．

たとえば，文字列中で指定した文字の有無やその位置を調べる関数として，strchrやstrrchrがあります．これは，次の例のように使用します．

```
char *p, *string;
p = strchr (string, 'Z') ;
```

これは，文字列string中で最初に見つかった文字'Z'のアドレスをpに返します．しかし，strchrやstrrchrなどの標準の文字列操作関数では，2バイト文字のことをまったく考慮していないため，もしも全角文字の2バイト目に'Z'と同じ値があった場合，その位置を返してしまうことになります．したがって，日本語を含んだ文字列を正しく処理するためには，これらの標準の文字列操作関数は一般には使用できないことになります．そこで，両処理系とも，日本語処理を行うための文字列操作関数や文字関数をサポートしています．表3に，各日本語処理ライブラリ関数を示します．

基本的には，これらの日本語処理関数はマイクロソフト社がMicrosoft C 3.0日本版の発表時に仕様を作成し，各ベンダに仕様の共通化を提案したという経緯があるため，両処理系間

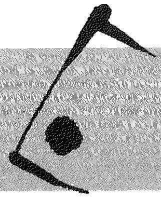
表2　PC-9801版BIOSライブラリ関数

BIOS	Microsoft C 6.0	Turbo C 2.0	Turbo C++ 1.0
ハードウェア情報	_bios_equiplist	bios98equip	pc98equip
タイマ		bios98stoptimer	pc98timer
		bios98timer	
カレンダクロック	_bios_timeofday	bios98time	pc98clock
キーボード	_bios_keybrd	bios98key	pc98fkeyon
			pc98fkeyoff
			pc98fkeychk
			pc98getfkey
			pc98key
			pc98setfkey
CRT画面			pc98crt
ディスク	_bios_disk	bios98disk	pc98disk
		bios98harddisk	
マウス		bios98mouse	pc98mouse
		bios98mouse_init	
プリンタ	_bios_printer	bios98print	pc98print
RS-232C	_bios_serialcom	bios98com	pc98comchkbuf
		bios98com_ch2	pc98comchkbufx
		bios98com_ch3	pc98comgetstat
		bios98com_init	pc98comgetstatx
		bios98com_init_ch2	pc98cominit
		bios98com_init_ch3	pc98cominitx
			pc98comrecv
			pc98comrecvx
			pc98comreset
			pc98comresetx
			pc98comsend
			pc98comsendx
			pc98comsendbreak
			pc98comsendbreakx
			pc98comsetstat
			pc98comsetstatx
			pc98comterm
			pc98comtermx
メモリサイズ	_bios_memsize	bios98memory	pc98mem
メモリスイッチ		bios98msw	
フォント		getfont	pc98getfont
		putuserfont	pc98putuserfont

ワンラインNEWS
日立マクセルは特許電子出願用の3.5インチおよび5インチのフロッピーディスクを発売した．JIS規格で初期化されているので，出願時の手間が省ける．

ではほぼ完全に互換が取れています．ただし，Microsoft C 6.0では仕様の拡張を行い，関数をいくつか追加していますが，Turbo C++ 1.0ではまだそれに追従していません．

まとめ
機種／処理系の仕様の差異を熟知することが重要



MS-DOS上のC処理系，とくにMicrosoft CとTurbo Cについて，コンパイラのコード生成やライブラリの互換性という観点から両処理系の特徴を細かく比較検討してみましたが，いかがだったでしょうか？　これまで述べてきたように，MS-DOS上で実用的なアプリケーションプログラムを作成するかぎり，どうしても機種依存性や，処理系依存性はつきまとってしまいます．こういった依存性は必要悪ではありますが，できるかぎり少なくするように心がけることも大切です．そういう意味で，機種ごとや処理系ごとの細かい仕様の差異を熟知することはたいへん有意義だと思います．

この連載を続けている間に，Borland社からTurbo C++の後継としてBorland C++が発表されました．Turbo C++に対して細かい改良がかなり行われていますが，いちばん大きな特徴はMS-Windowsアプリケーションの開発をサポートしたことでしょう．今後は，MS-DOS上とはいっても，Windowsが標準的なアプリケーションのプラットフォームになってくると思われます．Windows上の場合，原則的に機種依存性はなくなりますが，やはり各言語処理系の特徴を細かく把握しておくことの重要性は変わらないと思います．この記事が，今後も少しは読者のみなさんのお役にたてば，筆者としてはたいへん嬉しく思います．

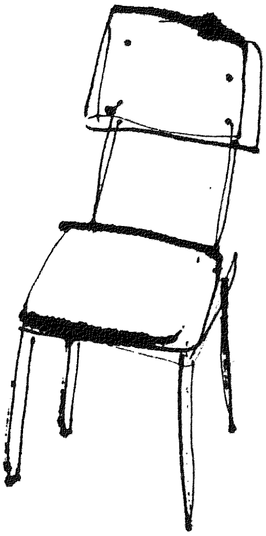


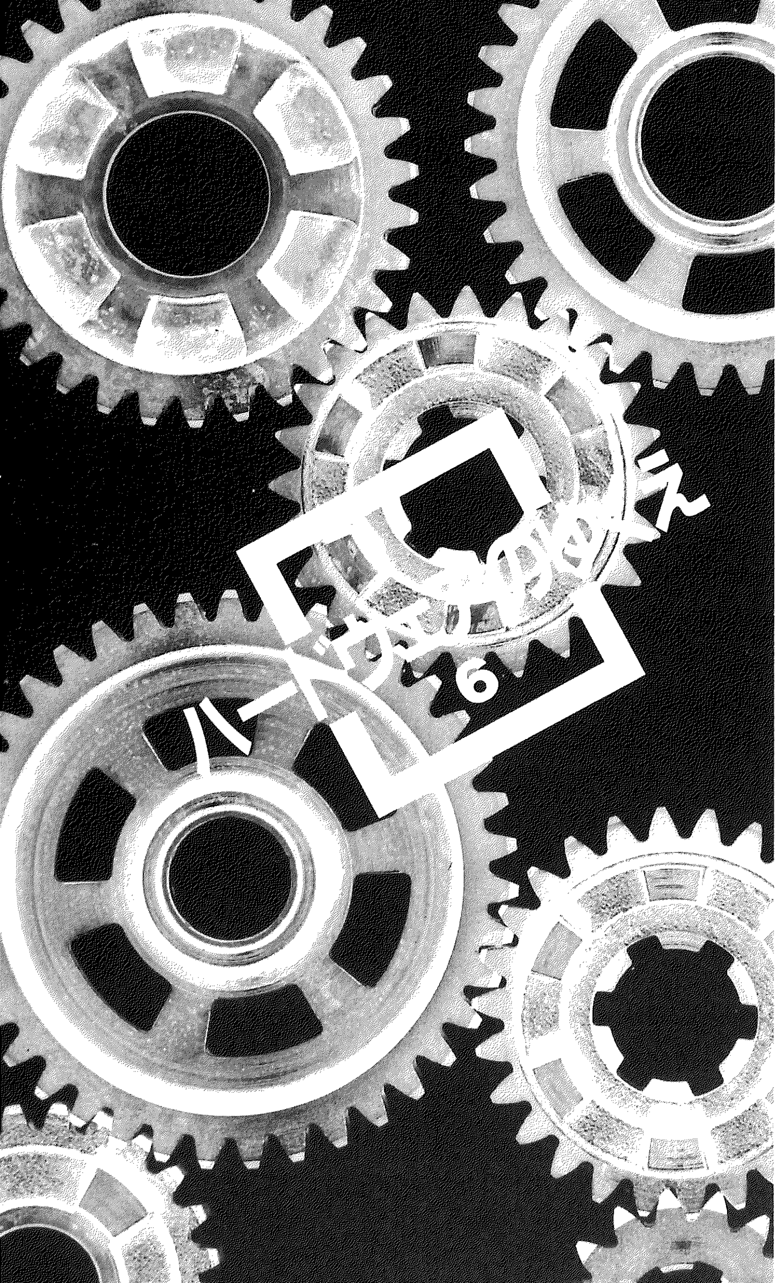
表3　日本語処理ライブラリ関数

	MS-C 6.0	MS-C 5.1	TC++	TC 2.0
btom	○	○	○	○
chkctype	○	○	○	○
_fbtom	○	－	－	－
_fjstr系関数	○	－	－	－
_fmtob	○	－	－	－
_fnthctype	○	－	－	－
hantozen	○	○	○	○
isalkana	○	○	○	○
isalnmkana	○	○	○	○
isgrkana	○	○	○	○
iskana	○	○	○	○
iskanji	○	○	○	○
iskanji2	○	○	○	○
iskmoji	○	○	○	○
iskpun	○	○	○	○
ispnkana	○	○	○	○
isprkana	○	○	○	○
jasctime	－	－	－	○
jctime	－	－	－	○
jisalpha	○	○	○	○
jisdigit	○	○	○	○
jishira	○	○	○	○
jiskana	－	－	○	○
jiskata	○	○	○	○
jiskigou	○	○	○	○
jisl0	○	○	○	○
jisl1	○	○	○	○
jisl2	○	○	○	○
jislower	○	○	○	○
jisprint	○	○	－	－
jisspace	○	○	○	○
jistojms	○	○	○	○
jisupper	○	○	○	○
jiszen	○	○	○	○
jmstojis	○	○	○	○
jstradv	○	○	○	○
jstrchr	○	○	○	○
jstrcmp	○	○	○	○
jstrcspn	○	－	－	－
jstricmp	○	－	－	－
jstrlen	○	○	○	○
jstrlwr	○	－	－	－
jstrmatch	○	○	○	○
jstrncat	○	○	○	○
jstrncmp	○	○	○	○
jstrncpy	○	○	○	○
jstrnicmp	○	－	－	－
jstrnset	○	－	－	－
jstrrchr	○	○	○	○
jstrrev	○	○	○	○
jstrset	○	－	－	－
jstrskip	○	○	○	○
jstrspn	○	－	－	－
jstrstr	○	○	○	○
jstrtok	○	○	○	○
jstrupr	○	－	－	－
jtohira	○	○	○	○
jtokana	－	－	○	○
jtokata	○	○	○	○
jtolower	○	○	○	○
jtoupper	○	○	○	○
mtob	○	○	○	○
nthctype	○	○	○	○
zentohan	○	○	○	○

Reintroduction to HARDWARES

プログラマのための
ハードウェア再入門

鎌田正寿



コンピュータシステムのデータ処理能力は年々着実に上がっています．マイクロプロセッサをCPUとするコンピュータシステムを考えてみても，CPUが一度に扱えるデータ量も8ビット，16ビット，32ビットと増え，CPUに与えるクロックも2MHz程度から50MHz近くまで上がり，複数クロックで一命令を実行していたものが，パイプライン化などの技術を用いて1クロックで一命令が実行できるCPUが増えてきました．いまあるCPUの延長上のままで，どこまで性能を上げることができるのでしょうか．将来どのようなCPUがでてくるのでしょうか．

CPUのゆくえ

最初に具体的な数値を出すための前提条件をつけておきます．それは通常の基板サイズ１枚で，システムの主要な部分を構成できることです．スーパーコンピュータや大型汎用機のように個人では絶対に買えない価格，大きさのシステムはここでは想定していません．

・動作クロック

たとえば，100MHzのクロックは10nsが１周期となります．10nsというと，光の速度で３m進む計算になります．これは理論上での限界であるというだけで，現実のデバイスでは信号のたちあがり時間などがありますからもっと制限はきつくなります．

動作クロックが上がった場合に，いちばん問題になるのはメモリへのアクセスです．現在，一般的なDRAMでは80ns程度のアクセススピードであり，10nsとは８倍の開きがあります．この差を埋めるためにキャッシュメモリが使われますが，このキャッシュメモリを構成するSRAMも高速で大量に，しかも安価にとなると20ns以下のものはコスト面でかなり苦しくなります．これを解決するためにCPU内部にキャッシュメモリを内蔵することにしてもオンチップでは容量が増やせません．

・ビット長

プログラムで一度に扱うデータのビット長が，CPUのビット長よりも大きい場合には，ビット長を増やすことで処理能力は確実に上がりました．しかし，倍精度の浮動小数も64ビットですから128，256ビットと単純に上げても，同じような効果は望めないでしょう．レジスタとデータバスとアドレスバスが完全に64ビットになれば，それ以上は配線の複雑化などでデメリットのほうが多くなるように思われます．

クロックとビット長などからすると，近い将来，現在と同じような処理方法で実現可能なCPU性能は５倍程度が限界ではないでしょうか．その次の展開として，すぐに思いつくのはマルチCPU化ですがこのマルチCPUにはどのような問題があるのでしょう．

・マルチCPUの結合方法

単純な方法では，共有メモリを使い結合します．この方法では，CPUの数が増えると排他処理のためにまたされる現象が多くなります．キャッシュメモリを併用し，共有メモリへのアクセス頻度を減らしたとしても，4CPU，多くても8CPUが限界です．

CPUをアレイ状に並べたり，ハイパーキューブのように並べる方法では，配線のための面積が多く必要で，システム全体も複雑化を避けることはできません．

・ソフトウェアの互換性

マルチプロセスのOSならば，プロセスごとにCPUを割り当てるような対応は比較的簡単にできます．この場合は，従来のソフトはアルゴリズムを変更しなくてもそのまま動きます．しかしこの方法では，システム全体の処理能力を上げることができたとしても，単一のプログラムの処理能力は上がりません．

単一のプログラムの処理能力を上げるために，ループ内のそれぞれの処理を複数のCPUにふり分けるようなコンパイラの作成も可能ですが，まだこのような技術は確定したものではありませんし，部分的にしか処理をふり分けることができないはずです．

完全に並列思考で最初からプログラムを作るならば，もっと多くの範囲を並列化できますが，この場合はアルゴリズムそのものから従来の処理とは変更する必要があります（つまり従来のソフトを捨てる必要があります）．

マルチCPU化では，このようにフルに性能を生かそうとすれば，ソフトウェアをすべてアルゴリズムから作りなおす必要があり，ハードウェアもCPUの数に比例して複雑になってしまいます．

・単純なCPUの可能性

一枚の基板でシステムを構成しようとするならば，現在のCPUと同じ規模のCPUの数を増やすのはおそらく現実的ではなく，もっと単純なCPUを大量に接続するような構成でないと無理ではないでしょうか．理想的には，現在のDRAMと同じくらいのピン数でメモリなどはすべて内蔵し，CPUの外との通信方法も，ピン数が少なくてもよいようなものに期待したいのですが，そのようなCPUは聞きませんね．

トランスピュータというCPUが最初でてきたときは，そのような方向に進むのだろうかと期待したのですが，どうも違うようで，現在の規模は普通のCPUと同じです．

そのようなCPUがでてきていないのは，並列処理そのものがまだ技術的

に十分確立していないからでしょう．このため技術的にも採算上でもリスクが大きく，研究段階の域を脱していないのではないでしょうか．

ソフトウェアに近いハードウェア

ちょっと視点を変えて，ソフトウェアとハードウェアの本質的な違いについて考えてみましょう．

・開発方法

仕様を検討し設計を行うときには，ソフトウェアもハードウェアもそれほど本質的な違いはないはずです．実際に試作を行うなどの作成作業になると，ソフトウェアはコード化されたデータというソースコードやオブジェクトコードを扱うだけですみます．マシンの前に座ってキーボードで操作するだけで「モノ」はできてしまいます．

それに対してハードウェアは，CADやシミュレーションソフトを使ったとしても，実際の「モノ」は部品を調達し組み立てる必要があります．

・製造方法

ソフトウェアでは，もとになるソースコードからオブジェクトコードを作るにはコンパイラやアセンブラを使いますが，これは長くても数分で終わることです．これに対してハードウェアを実際に作るまでには，回路が決定されてから実際の基板ができてくるまでに，早くても数週間はかかります．

製造が試作段階で数が少ない場合には，基板を作らずにラッピングなどで作ることもできますが，最近の高速で複雑なデバイスに対応するには，動作の安定性やラッピングする作業時間を考慮すると，基板を作るほうが結局は早くて確実であるような状況になっているようです．

ソフトウェアの場合には，同じものを作るにはコピーするだけで終わりです．複数のマシンでまったく同じソフトウェアを動かすのは，ごく単純な作業だけで確実にできるのです．それに対してハードウェアの場合は，1つずつ同じ作業が必要になります．

・改変方法

ソフトウェアにしてもハードウェアにしても，最初に作ったものが完璧でバグがないというのはまずあり得ません．もし，それ自体は問題がない状態であっても，機能向上とかそのシステム以外の要因への対応などで，改変することは頻繁に発生します．このとき，改変部分をどうすべきかという検討をする段階では，ソフトウェアとハードウェアの違いはそれほどありませんが，実際の改変を行うには製造方法と同じ手間がかかります．

改変がごく一部である場合には，最初から全部作りなおさずに，その部分だけに対応するという方法もあり，ソフトウェアでは「パッチをあてる」という作業に，ハードウェアでは「カットする」または「ジャンパーをとばす」という作業になります．とはいってもソフトウェアのパッチは最近ではコンパイル／アセンブルの時間が短いですから，以前にくらべれば使われることはあまりありませんね．

なぜ，ソフトウェアとハードウェアの違いについて書いたかというと，「よりソフトウェアに近いハードウェアを作るにはどうしたらよいか」ということを考えているからです．もし全部ではないにしても，ハードウェアがもっとソフトウェアと同じように扱えるなら，ハードウェアの開発・製造時間が短く，そしてコストも下がるはずだからです．開発・製造時間が短くそして作成が簡単になれば，もっと多様化されたハードウェアを使えるようになるでしょうし，多様化が簡単になれば，それぞれのシステムでの最適化も進み性能も上がるはずです．コストが下がれば，同じ費用でより多くのロジックをシステムに入れることができます．

そのためには何が必要なのかというと，ハードウェアの個々の構成要素が特定の用途だけに使われるのではなくて汎用化されることです．ソフトウェアが動作するときに，その実体はメモリ上に配置されます．メモリそのものはアドレスや内容は違いますが，構成要素としてみれば同じものからできています．それと同様に，ハードウェアも単一の構成要素を基本として作成することができれば，いちいち基板を作ったりハンダづけする仕事が減るでしょう．

部分的には，このようなプログラム次第で動作を変えるハードウェアはすでに存在しています．PAL，GAL，LCAとか，プログラマブルゲートアレイと呼ばれるものがそうです．ダイナミックに内容を変更できるものはLCA（Logic Cell Array）ですが，まだ現在あるデバイスでは1，2万ゲート程度であり，システムの一部に使う用途を目的としています．しかし近い将来には，このようなデバイスだけでシステムのほとんどを構成すること

も不可能ではなくなるでしょう.

ダイナミックなハードウェア

完全にプログラマブルでダイナミックに構成を変更できるハードウェアだけで，マシンを作ることができたら何ができるのでしょうか.

・マシンの違いは基本クロックと内蔵する素子の数の違い

細かくいえば，I/F用のコネクタの形状とか，電源関係とか，基板の形状なども違いがあるでしょうが，これは本質的には関係ありませんよね. いま，CPUが違ったりマシンが違うためにソフトウェアの互換性がないという事態の多くは，OSを変更する程度の手間で対応できるはずです.

・実行するソフトウェアにそのつど最適化したハードを構成できる

命令セットそのものもすべて変更可能ですから，たとえば，特定の数値計算だけ高速化したい場合には，それに適した命令セットまでコンパイラで生成してしまえばいいのです. 命令セットまでとはいっても，それを実行するハードウェアも最適化したものを生成できるのですから，かなりおもしろいことが可能なはずです.

もちろんデメリットもあります.

・動作速度が遅い

プログラマブルなチップだと，プログラマブルではない通常のチップにくらべて，チップ内部の配線が最短距離を通ることができず，自由度も少ないので，どうしても動作速度が遅くなる. ダイナミックという自由度を得るために，固定の場合の配線の自由度が下がるわけです.

・集積度が低い

これもプログラマブルであるがために，汎用性と引換えに専用で作られたチップにくらべれば必ず集積度は下がります.

もしこのようなマシンを作ることができたとしても，おそらく一般には広まらないでしょうね. でも将来のマシンのひとつとして，このようなマシンも選択できれば単に開発用や研究実験用だとしても有用だと思うのですが，どうでしょう？　また全部がプログラマブルでないとしても，部分的に主要な制御部分だけでもこうなっていれば便利でおもしろいと思います.

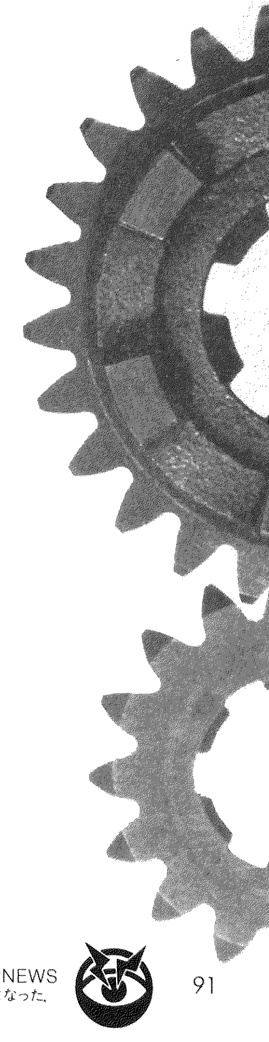
ずっとずっと将来のマシンは，たぶんこのようなダイナミックな構成を基本にすると予想してます. そうでないとすると，並列処理のところで書いたごく小さいCPU要素をチップ上に多数載せたような感じでしょうか.

チップ上に多数載せた場合の問題は，ウエハーから取れる歩留まりでしょうが，これは構成要素が(DRAM／SRAMみたいに)同じ繰り返しであるなら，不良の部分を避けて使えば解決できるはずです. ちょうどハードディスクのディフェクト処理のように.

最後に

いままで6回にわたって,「プログラマのためのハードウェア再入門」という題で連載をしてきましたが，今回で終了とさせていただきます（正直いってネタが切れたのです). (^_^;

このような雑誌の記事を書くというのは，実をいうと初体験で，文章を書くのも苦手なので苦労しましたが，その割には内容がたいしたことがなくてごめんなさい（_ _). でも，ハードウェアってソフトウェアと同様におもしろいですよ. みなさんも機会があったら（なくてもかな？）恐れずにハードウェアにふれてみてくださいね.



MS-DOS TOOL LABO

MS-DOSツール・ラボ

第3回 再帰と式の評価 簡易16進電卓 EXPR.EXE

加野島英渡

プログラムを作成する場合にもっとも頭を悩ますのは，やはりどのようなアルゴリズムを用いるかということだろう．ちょっとしたツールを作る際にもこの点は重要である．今回は，ディレクトリツリーをたどりながら，同じ動作を繰り返すようなツール作成に有効な，「再帰」について解説するとともに簡易16進電卓を作成する．

再帰する話

昔あるところに男が1人おってな，さくらんぼを食ったときにもったいないからと種まで飲み込んでしまったそうな．しばらくすると頭が妙にむずがゆい．恐る恐る触ってみると，なんと飲み込んださくらんぼの種から芽が出ておった．こいつは珍しいわいとしばらく放っておくと，みるみる伸びてきて春にはりっぱに桜の花が咲いた．

さぁ近所の連中が花見だというので，番茶だのたくわんだのもち寄ってわいわいと大騒ぎ．夜は夜とて夜桜見物に集まるので頭が騒がしくて寝ておられん．たまりかねて男は頭にはえたさくらの木を引っこ抜いてしもうた．

やれうれしや，これでゆっくり眠れるわいと喜んだが，その夜の大雨で引っこ抜いたあとの穴に水がたまって池になった．こんどはその池に釣りだ舟遊びだとまた人が集まる．おまけにガキどもがおもしろがってドボンドボンと飛び込むので，うるさくてこれまた寝ておられん．哀れな男はとうとう池に身をなげてしもうたそうな．

再帰ってなんだろう

筋立てはうろ覚えながら，古典落語の「あたま山」からいただいた．この方面に詳しい方がおられたら，笑って見逃してほしい．

さて「あたま山」では，男が池に身をなげてめでたしめでたしとなるわけだが(めでたくなんかないな，こりゃ)，ようく考えるとちょっと変．だいたいこの池って自分の頭にできた池じゃないか．その池に自分が身をなげる？その，身をなげる男の頭に池があるん

じゃないか．だから池に身をなげようとしている男の頭には，その池に身をなげようとしている男がいて……いや，よそうよそう．これを考えだすとおかしくなってしまう．

われわれの身の回りには，このようにあるものの中にそれ自身を含んでいるようなもの，あるいはそういう現象がある．最近では見かけないが，子供向けの番組で，テレビの前に立っている人をテレビカメラで映すというのがあった．テレビカメラで映した画像はそのテレビに映しだされるようになっている．こうするとテレビカメラで映した画像がテレビのほうに再現され，それをまたテレビカメラで映すので，映っている人間（とテレビ）が段々小さくなりながら無限に続いて映っているように見える．

算数のほうでも，このようにあるものの中にそれ自身を含んでいるという話が出てくる．よくひき合いに出されるのは「階乗」．階乗の定義というのは，「nの階乗とは，nとn－1の階乗をかけたものである」となっている．階乗というものの説明をしているのに，その説明の中にまだ説明の終わっていない階乗というものが参照されているので，話が堂々巡りをしているような変な気分になる．しかしこれは堂々巡りではないのだ．

フィボナッチの数列というのもある．知ったかぶりをするつもりはないのでさっさと白状してしまうが，だいぶ前に読んだ本の受け売り．「1対のウサギがいて，1か月たったらそのあとずっと1対ずつのウサギを産むとしたら，nか月後には何対のウサギがいるだろう」という話だ．これを考えだしたのがフィボナッチという人なのでこれをフィボナッチの数列というそうだが，この話を聞いたときには，「名前だけでなく考えることも変わってるなぁ」と思ったものだ．

で，このフィボナッチの数列だが，「nか月目のウサギの数は，n－1か月目のウサギの数とn－2か月目のウサギの数を加えたもの」となっている．これもまた，自分自身の定義の中に自分自身を含んでいる．実際にプログラムを書いてみるとおもしろいが，話を進めよう．

自分の中に自分自身を含む，それが再帰

むりやり話をコンピュータのほうにもってきてしまうが，説明しているうちに話がもとに戻ってしまって困るものはコンピュータの世界にもある．

Q：サブディレクトリとは？
A：ファイルかサブディレクトリが集まったもの．（図1）
（だから，そのサブディレクトリってなんなんだ）

このように，定義の中に自分自身を含むものを再帰的な定義という．まだある．式の定義などこれの代表格だ．

Q：式とは？
A：単純式であるか，単純式同士の関係演算である（>, <, =）
Q：単純式とは？
A：項であるか，項同士の加減演算である（+, −）
Q：項とは？
A：因子であるか，因子同士の乗除算である（*，／）
Q：因子とは？
A：定数であるか，式をカッコで括ったものである（？？？）
（だから，その式のことを聞いてるんだ）

言葉だとちょっとわかりにくいので，Pascalでおなじみの構文図（図2）というのを書いてみよう（余談だが，この構文図というのは別にPascalの専売特許ではない）．

非常に単純な式の定義であるが，因子とは定数であるか，もしくは式をカッコでくくったものということになって，式の定義自身に式というものを含んでいる．これも再帰的な定義だ．

さて，こういう定義をいくつか見ていくと，ある疑問にぶち当たる．そう，「終わらないじゃないか」という疑問だ．たとえば階乗だが，定義はこれで正しいとしても，もしこれを解くプログラムをこのまま書くといつまでたっても終わらないことになる．いつまでたっても階乗の計算プログラムは，与えられた数から1つ引いたものの階乗を計算しつづけることになる．際限なく自分自身を呼び出すことになり，話としてはおもしろいが，これでは少し困る．

落語の世界の話なら頭の上に身投げ候補が何人いてもいいだろうし，テレビの中に何人の小人が住んでいようと別段困りはしない．しかし問題解決のために使われるプログラムは，ともかくなんらかの答えを出さなくてはならない．いつまでも再帰していて返ってこないのでは困ってしまうのだ．これ

図1　サブディレクトリの例

```
¥ ------+----- subdir1
        +----- file1
        +----- file2
        +----- subdir2 -+------ file2-1
                        +------ file2-2
                        +------ file2-3
                        +------ subdir2-1 ------+----- file2-1-1
                                                +----- file2-1-2
```

を再帰（再起）不能という……なんてばかなことをいっている場合ではない．

再帰するプログラム

プログラムの中から自分自身を呼び出すような構造のものを「再帰的なプログラム」と呼んでいる．再帰的な問題を解くには，プログラム自身もまた再帰的にできているほうが都合がよろしい．階乗の定義が再帰的であるならば，階乗を計算するプログラムもまた再帰的な構造をとり，再帰的な動作をするのが自然である．

そこで前節の問題．戻ってこないプログラムでは困る．戻ってこないだけならいいが（よくはないけど），困ったことに自分を呼び出すときにスタックを消費するので，しまいにはどこか大事なところを壊して暴走する可能性もある．

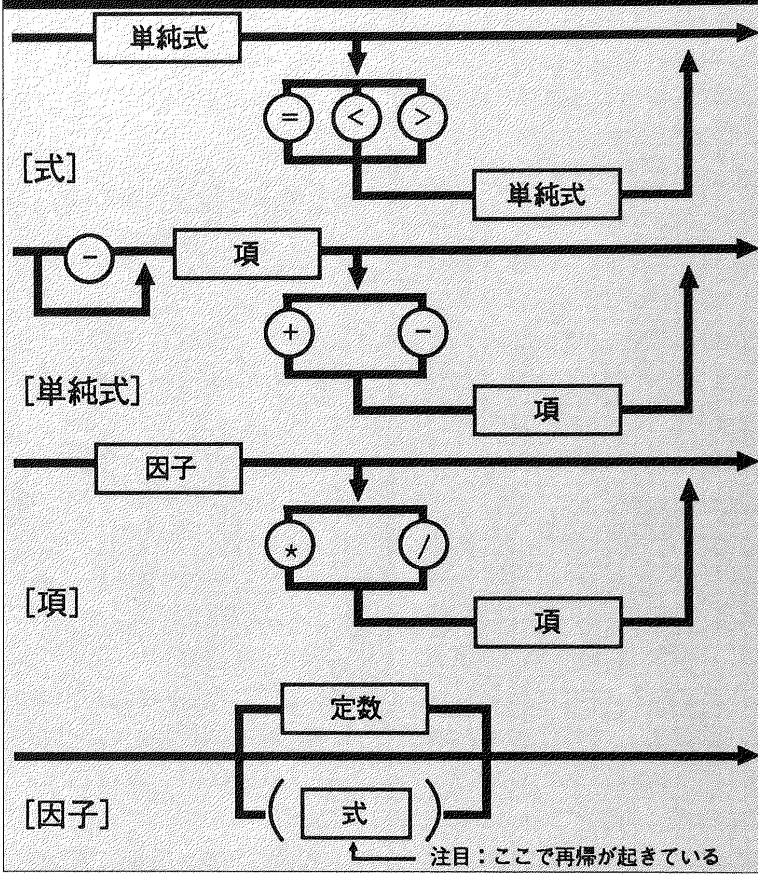
こういう状態を無限再帰と呼んでいるのだが，無限再帰にならないためには，再帰を打ち切るための条件が必要だ．「これこれの条件の場合は再帰せずに（決まった結果を返して）実行を終わる」というロジックがなければ，再帰するプログラムは永遠に終わらない．

じつはこの説明のためにわざと書かなかったのだが，上の階乗のロジックには「0の階乗は1である」というのが抜けている．これが階乗のロジックの終了条件だ．したがって，プログラムは引数として0が与えられたらそれ以上自分自身を呼び出すことをやめて，1を結果として呼び出されたプログラム（これが自分自身だからややこしいのだが）に戻ればいい．

ためしに，階乗を解くプログラムをアセンブラで書いてみよう（リスト1）．あたま山のアセンブラ版だ．symdebで追いやすいようにあたまにメインルーチンをつけておいたから，(1) のところにブレークポイントを置き，AXに適当な値を入れてmainを呼び出してためしてみてほしい．

実行結果を見ても，まだだまされているような気のする人がいるかもしれない．そういう場合は，ためしにAXに3くらいの値が入っているとして，実

図2　構文図



リスト1　kaijo.asm

```
_TEXT     SEGMENT
          ASSUME  CS: _TEXT , SS: _STACK
;
main      proc
          call    fact
          ret             ; ---------------------------- (1)
fact:
          cmp     ax,0    ; もし渡された値が0ならば
          jnz     fact_0
          mov     ax,1    ; 1を返して終わり
          ret
fact_0:
          push    ax      ; たとえばここの値が3なら
          dec     ax      ; それから1つ引いて2にして
          call    fact    ; 自分自身を呼び出し
          pop     bx      ; AXに入ってきた結果と
          mul     bx      ; さっきの値（3）を掛けて
          ret             ; その値を持って帰る
;
main      endp
;
_TEXT     ENDS

_STACK    SEGMENT STACK
          db      1024 dup (?)
_STACK    ENDS

          END     main
```


ワンラインNEWS
オムロンは米大手ミニコンピュータメーカー「データ・ゼネラル社」の日本法人「日本・データゼネラル」を買収することで合意した．

際にプログラムの動きを目で追ってみるといい．小さな値を与えるのは，大きな値だと最後までいくのに時間がかかってしまって途中でいやになってしまうからだ．

このとき，自分自身を呼び出すところでわけがわからなくなってしまうことが多いので，与える数＋１個のサブルーチンをエディタのコピーコマンドでコピーしておく．与える数が２なら，３個のまったくおなじサブルーチンを作っておいて，アセンブルする必要などないからこのままプリントアウトしておく．そうして，自分自身を呼び出すところで，次の（下の）サブルーチンを呼び出すものとして追っていくわけだ．こうするとよくわかる．

最後まで追いきれたかな？

図３　kaijo.asmの実行結果

```
-r
AX=0000  BX=0000  CX=0016  DX=0000  SP=0400  BP=0000  SI=0000  DI=0000

-g=0,3           ; AX = 0なら結果は1
AX=0001  BX=0000  CX=0016  DX=0000  SP=0400  BP=0000  SI=0000  DI=0000

-g=0,3           ; AX = 1でも結果はおなじ
AX=0001  BX=0001  CX=0016  DX=0000  SP=0400  BP=0000  SI=0000  DI=0000

-r ax
AX 0001
:3
-g=0,3           ; AX = 3なら結果は6 (3! = 6)
AX=0006  BX=0003  CX=0016  DX=0000  SP=0400  BP=0000  SI=0000  DI=0000

-r ax
AX 0006
:8
-g=0,3           ; AX = 8なら結果は40320(9D80H)
AX=9D80  BX=0008  CX=0016  DX=0000  SP=0400  BP=0000  SI=0000  DI=0000
```

リスト２　kaijo.asmの変更

```
tmpvar  dw      (?)

fact:
        cmp     ax,0    ; もし渡された値が0ならば
        jnz     fact_0
        mov     ax,1    ; 1を返して終わり
        ret
fact_0:
        mov     tmpvar,ax       ; push  ax
        dec     ax              ;
        call    fact            ;
        mov     bx,tmpvar       ; pop   bx
        mul     bx              ;
        ret                     ;
```

再帰するプログラムの注意点

さて，このような再帰的なロジックをプログラミングする場合に，いくつか注意しておかなくてはならないことがある．必ず終了条件を作ること，というのはもう書いたから，それ以外のことについて書いておこう．

(1)内部で使う変数を静的に宣言してはいけない

再帰的なプログラムにかぎらないが，あるプログラムの中からなにかのサブルーチンを呼び出した場合，帰ってきたときに自分自身が使っている変数が変化していたのでは困ってしまう（もちろんそれを期待してプログラムを書いている場合は別だが）．

ものはためし，さっきのプログラムを静的な変数を使うように変更してみればいい（リスト２）．

わかりにくければ，さっきのようにこのプログラムを２つ３つコピーして追ってみるとよろしい．変数が壊れていることがよくわかるだろう．もちろん，変数はサブルーチンごとに用意されるのではなく，１つだけが宣言されているとして考えてもらいたい．

自分自身を呼び出す場合は，呼び出された自分自身は必ずおなじ変数を使うので，戻ってきたときにはまちがいなく変数の中身が変わってしまっている．これでは結果が保証できないことになるので，内部で使う変数は必ず動的な（プログラムの動作に応じてダイナミックにアドレスを与えられる，というような意味）ものでなければならない．

これにいちばん適したデータ構造はスタックである．スタックについては次回以降で説明しようと思っているが，スタックというのは，必ず現在のスタックポインタより低位（作り方によっては高位）番地の方向に使っていくようになっているので，これを底上げしてそこに変数領域を確保しておけばいい．

サンプルではそんなことはしていないが，これは内部で使う変数が１個だけなのでレジスタを変数として使い，呼び出すときにそれをスタックにPUSHすることで結果的にスタック上に変数領域をとったことにしているのだ．ためしにきちんとスタック上に変数をとったバージョンを作ってみようか（リスト３）．

ちょっとわかりにくいので，図に書いてみよう（図４）．

呼び出されるごとにスタック上に変数領域が確保され，おたがいに干渉しあわないことが理解できるだろう．Cなどの高水準言語では動的な変数を使用する場合には，宣言をするだけでこういう操作をする一連のコードが自動的に生成されて，再帰的なプログラムを書くのが容易になっている．

このように，スタック上に作られたデータ構造をスタックフレームなどと

呼んでいるのだが，スタックフレームの生成だの消去だの，こういうことをちゃんとやってくれるというのが高水準言語のわずらわしくもありがたいところ．Cコンパイラが生成したオブジェクトを見ていると，このスタックフレームをひんぱんにアクセスしているようすがよくわかる．このようすを「スタックフレームにまとわりつくような」と表現したものをどこかで読んことがあって，うまいことをいうなぁと感心してしまった．

もっとも，最近のアセンブラではこのへんはもう少しかんたんに書けるようになっているし，CPUも少しは賢くなってきて，enterだのleaveだのとどこかで聞いたような命令で前述のような操作を一気にやってしまおうというご時勢だ．こういった機能とごりごりのマクロを組み合わせ，インデントをつけて書かれたプログラムを見るととてもアセンブラとは思えない．そういうものもまた好きではあるけれど．おっと，話が脇道にそれてしまった．

(2)スタック領域は十分余裕をみておくこと

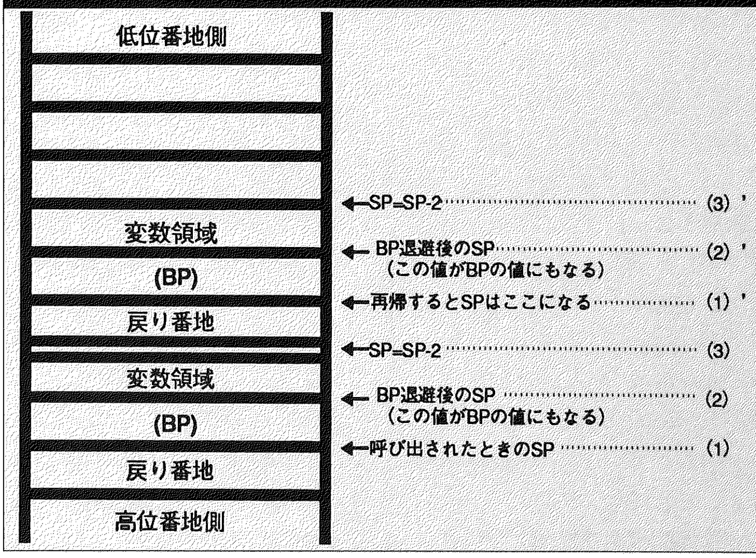
当然といえば当然な話．再帰するプログラムはスタック上に動的な変数を確保するうえ，自分自身を何重にも呼び出していくので，そのための戻り番地もまたスタックに積まれる．スタック領域の不足はすぐに現象があらわれないことがあるので要注意．運用段階に入ってから，ちょっと複雑なケースに当たったり大量のデータを処理した場合に出てくることが多い．

どれくらいの領域を与えればいいかはそのプログラムの構造によるのでなんともいえない……のであるが，「なんともいえない」くらいならだれでもいうので，かわりに泥臭いが確実な方法を1つ．スタック領域をあらかじめあるデータで埋めておく．00HでもいいしFFHでもいい．デバッガにかけている場合は，実行終了後にその領域をダンプしてみればどこまで使われたかがわかる．デバッガにかけにくい場合は，スタック領域を画面かファイルに出力するプログラムを実行終了直前に呼び出すようにしておく．

リスト3　スタック上に変数をとるバージョン

```
main    proc
        call    fact
        ret
fact:                           ; ................................... (1)
        cmp     ax,0
        jnz     fact_0
        mov     ax,1
        ret
fact_0:
        push    bp              ; ................................... (2)
        mov     bp,sp           ; ＢＰをポインタにして
        sub     sp,2            ; スタックを底上げして変数領域を確保... (3)
;
        mov     [bp-2],ax       ; 変数へのアクセス
        dec     ax
        call    fact
        mov     bx,[bp-2]       ; おなじく
        mul     bx
;
        mov     sp,bp           ; 変数領域を捨てる
        pop     bp
        ret
;
main    endp
```

図4　スタックの状態



こうしておいて，考えられる最悪のケースを与えて得られた最大消費量の倍を与えておけばまずは大丈夫だろう．最大消費量がわかったのなら，それだけでいいじゃないかと思われるかもしれないが，それじゃだめなのだ．トラブルというのは設計者の想定していなかった使い方をされた場合に起きるものなのだから．ものにもよるが，設計者の考えた最悪の条件なんて，ユーザーの考えるごく普通の使い方にすぎない．いつも財布とスタックには余裕をもちたいものだ．

(3)なんでもかんでも再帰でやろうと思わないこと

再帰というのは非常におもしろい考え方なので，はまる人ははまってしまう．単なるループで解決できるものでも無理矢理，再帰を使ってしまう人がいるようだが，どちらでも解決できる

問題ならばループのほうが効率がいい．再帰を使うと動的な変数の確保や自分自身の呼び出し／復帰によるオーバーヘッドがかかるので，実行時間は遅くなるし，スタック領域の消費も大きい．

最近のCPUは，スタックフレームの操作が楽になるような命令群が用意されているので，昔のCPUよりは再帰を効率よく処理できるが，それでもループに比べれば効率は劣る．ここいちばんというときに上手に使ってほしい．なお，その定義の中で自分自身を２度以上再帰的に呼び出すような構造のプログラムはループに変更するのが困難である，ということを覚えておいてほしい．

今月のプログラム

コマンドラインオプションで式を与えて，その値を評価して結果を表示するプログラムを書いてみよう．簡易16進電卓というところだ．電卓プログラムならワープロソフトのおまけに付いてきたり，フリーソフトウェアで常駐型のものなどがあるようだが，こちらは教材だから機能は期待しないでほしい．そのかわりソースつきだから自分で機能を追加することも自由にできる．

プログラム自体は，先に書いた構文図にしたがってすなおにコーディングしていけばできてしまう．力まかせにエイヤッと書いてしまえば，非力な筆者でも半日もあればおおまかなロジックはできあがる．こういうかんたんなロジックでも，実用的なものにしようとするとエラー処理がけっこう問題なのだが，今回は入力エラーがあってもとくに処置はしない．暴走しないという程度でお茶を濁しておく．

構文図には関係演算(‘<’, ‘>’, ‘=’)も書いてあるのだが，コマンドラインオプションから入力することにしたので使えない記号が２つもあり（‘<’, ‘>’), 泣く泣く断念した．プログラムの構造としては，TERMとおなじものをEXPRESSIONとしてSIMPLE EXPRESSIONにかぶせればいいので，ためしてみてもらいたい．たぶんTERMの部分をコピーしてラベルなどを変更するだけで使えるはずだから．

また，今回は再帰的なプログラムを紹介するのが目的であることと，幸い演算子がすべて１バイトなので簡略化してしまったが，ほんとうのところは語句解析ルーチンをちゃんと呼びだして，それが返してくる「これは演算子のなんとか」とか「これは数値のこれこれ」という情報を使用して動作すべきである．

それから，ここではたとえば‘＊’がきたときにはかけ算をするというようにその場で演算してしまっているが，この演算しているところを「そういう演算をするCPUのマシンコードを出力する」というふうに書きなおすと，コンパイラのコード生成部になる．だから，「式の計算」といわずに「式の評価」と呼んでいる．評価と同時に計算してしまってもいいし，マシンコードを出力してもいいし，なんならエラー検出だけをしてもいい．重要なのは「評価」ということ．

式の評価の方法はこのやり方だけではなく，スタックを使ったものもある．筆者と同じように学校で正式に習っていない人のために，いずれ機会があったら紹介したいと思っている．

COLUMN　単機能か多機能か

ツールにかぎらず，自分で仕様を考えてプログラムを作るときにいつも悩むのがこれ．

使いやすさからは絶対に単機能のほうがいい．とにかく機能が限定されているのだから，いろいろなオプションだの環境ファイルだのとめんどうなことがない．ただし，ある限定された目的にしか使えない．

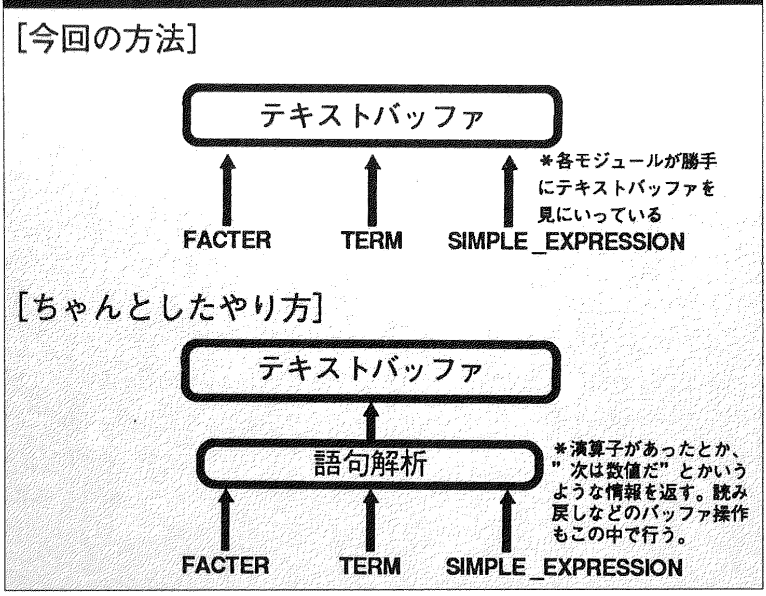
これに対して，多機能ソフトというのは１本でいろいろなことができる．ちょっとしたオプションの組み合わせや工夫で，さまざまな仕事をこなすことができるのだ．そのかわり使い方を覚えるのがたいへんで，マニュアルを開いて機能一覧を見ただけで降参してしまうこともある．

ここのところがなかなかむずかしい．最初のうちは単機能で使いやすいものを作るぞと決心してはじめるのだが，そのうち「最低限のオプションはやはり必要であろう」「いっしょにこんなこともできるといいな」てなことを考えだす．いつのまにか「こういうオプションはやはりあったほうがいい」「この機能があると絶対便利だ」になり，気がついてみたらドキュメントの半分がオプションの説明になってしまう．

それはそれなりに便利ではあるのだが，作者自身がいつも使っているのはその中のせいぜい２つか３つの機能だというのはめずらしいことではない．２つのプログラムを組み合わせればもっと柔軟に処理ができるものを，無理矢理ひとつのプログラムにつめこんではいないだろうか．

機能追加をする前に，この機能はほんとうに必要か，この機能はこのプログラムの中にあることが妥当かということを，よく考える癖をつけたいものだ．

図5　プログラムの実現方法



動作説明

では，いいわけが終わったので動作説明に移ろう.「式の評価」なんてカッコいいけど難しそう．どんなふうにしているんだろう……なんて期待していた読者には申しわけないが，ごらんのように実にあっさりとしたものである.

「動作は構文図のとおり」では説明にならないから，順を追って見ていくことにしよう．ちゃんと読んでいないと途中でだまされるかもしれない.

●メインルーチン

メインルーチンはとくにむずかしいところはない．例によって，コマンドラインオプションを自分自身のデータセグメントにコピーして管理しやすくしているだけ．コピーしたコマンドラインオプション(式が記述されている)の先頭アドレスをテキストポインタ(SI)にセットして式評価ルーチンを呼び出し，帰ってきたら戻り値を画面に表示する.

●単純式の評価

264行－277行

加減算と単項のマイナスの評価である.空白以外の最初の文字が‘－’であれば，その次を単純式と見て評価をしたあとで負数にする．なお，この式評価ルーチンでは単項のマイナスは式の先頭（ということはカッコの直後でもいい．カッコの中は，式であるということを思い出してほしい）にしかあらわれないという想定をしている．マイナス記号は，２つの項に対して作用する二項演算子としても使われるので，負数をあらわすために用いられる場合は「単項の」といっておかないと紛らわしくなる.

●単純式の評価の実体

233行－260行

まず項を評価する．項のあとが‘＋’でも‘－’でもなかった場合は,その項を返す.これらの演算子があった場合は,その次にあるはずの項を評価してから演算子に基づいて演算をした結果を返す.

●項の評価

200行－229行

まず因子を評価する．因子のあとが‘＊’でも‘／’でもなかった場合は，その因子を返す．これらの演算子があった場合は，その次にあるはずの因子を評価してから演算子に基づいて演算をした結果を返す.

●因子の評価

178行－196行

空白以外の最初の文字が‘(’であればこの中に式が書かれているので，単純式の評価をする（注意：ここで再帰が起きている).そうでなければ数値であるとして16進数値の評価をする.

●アセンブルとリンクの方法

A> MASM EXPR;

A> LINK EXPR+CONSOLE;

※〝CONSOLE.OBJ〟,〝CONSOLE.H〟,〝DOSFUNC.H〟は先月号のものと同じ.

●使い方

A> EXPR (800+4) ＊2

というように，コマンドラインから式を与えて起動すると結果を表示する.式の中で使える数値は16進のみで，結果も16進表示である．オーバーフローはそのまま切り捨てられるので注意すること．プログラムの構造が理解できたら，コマンドラインではなくMS-DOSの１行入力のファンクションコールを使って式を入力するようにしてみるといい．また，10進の数値も扱えるようにしたり演算子を増やしたりといろいろ工夫してみるとおもしろいだろう.

リスト4 expr.asm

```
TITLE   Command EXPR.EXE

PAGE    60,128

.xlist
include         console.h
include         dosfunc.h
.list

_TEXT   SEGMENT WORD PUBLIC 'CODE'
_TEXT   ENDS

_DATA   SEGMENT WORD PUBLIC 'DATA'
_DATA   ENDS

_STACK  SEGMENT STACK
_STACK  ENDS

DGROUP  GROUP   _DATA,_STACK
        ASSUME DS: DGROUP, SS: DGROUP
;
;
; ======================== 定数宣言 ================
;
PSP_PARAM       equ     80h             ; PSPのコマンドラインオプション
                                        ; のオフセット
;
NULL            equ     00h             ; ASCIZ文字列の終端子
;
; =================================================
;
_TEXT      SEGMENT
        ASSUME  CS: _TEXT
;
extrn           printmes:near,putchar:near,print_asciz:near
;
; -------------------- 十六進文字列をバイナリ値に変換
;
; 入力レジスタ  DS:SI=処理する文字列の先頭アドレス
; 出力レジスタ  DS:SI=最後に変換したデータのつぎのアドレス
;               BX=変換した値
;               CY:0=変換成功,CY:1=変換失敗
;
get_hvalue      proc
        push    ax
        push    cx
        mov     bx,0            ; まず結果を0にし
        mov     cx,bx           ; 処理個数をリセットする。
gth_lp:
        mov     al,[si]
        cmp     al,'a'          ; 小文字－>大文字変換
        jb      gth0
        cmp     al,'f'
        ja      gth0
        and     al,11011111b    ; 'a'-'f' --> 'A'-'F'
gth0:
        sub     al,'0'
        jc      gth_end         ; '0'より小さいのは数字ではない
        cmp     al,9            ; '0'を引いて9までなら数字
        jbe     gth_next
        sub     al,7            ; '9' ～ 'A'
        cmp     al,0ah          ; 10より小さければ数字ではない
        jb      gth_end
        cmp     al,0fh          ; 15以下ならば数字である
        jbe     gth_next
gth_end:
        cmp     cx,1            ; 数字がひとつもなければエラー
gth_end0:
        pop     cx
        pop     ax
        ret
gth_next:
        cmp     bh,10h
        cmc
        jc      gth_end0
        shl     bx,1            ; 4ビット左シフト（16倍）
        shl     bx,1
        shl     bx,1
        shl     bx,1
        or      bl,al           ; 結果に加えて
        inc     si              ; ポインタと
        inc     cx              ; カウンタを更新して
        jmp     gth_lp          ; もう1度
get_hvalue      endp
;
; -------------------- 数値を4桁の十六進文字列に変換
;
; 入力レジスタ  ES:DI=文字列を格納する先頭アドレス
;               AX=変換する値
; 出力レジスタ  ES:DI=最後に格納した文字列のつぎのアドレス
; 破壊レジスタ  AX
;
conv_hword      proc
        xchg    al,ah
        call    conv_hbyte      ; CALLが2回必要なところを
        xchg    al,ah           ; 1回ですませてしまう
conv_hbyte:
        push    ax
        shr     al,1
        shr     al,1
        shr     al,1
        shr     al,1
        call    conv_hnibble    ; ここもそう
        pop     ax
        and     al,0fh
conv_hnibble:
        add     al,'0'
        cmp     al,'9'
        jbe     cvh0
        add     al,7
cvh0:
        stosb
        ret                             ; RETはここ1か所だけ!!
conv_hword      endp
;
; -------------------- 表示バッファの出力処理
;
; 入力レジスタ  なし
; 出力レジスタ  なし
;
flush_buffer    proc
        push    si
        mov     si,offset dump_buffer
        call    print_asciz
        PRINTM  H_STDOUT,<0dh,0ah>
        pop     di
        ret
flush_buffer    endp
;
; -------------------- 式エラー処理
;
exp_error       proc
        PRINTM  H_STDOUT,<"EXP_ERROR :">
        call    print_asciz     ; テキストの残りを表示し
        PRINTM  H_STDOUT,<0dh,0ah>
        mov     sp,initial_sp   ; スタックを戻して
        mov     al,1
        jmp     exp_exit        ; 戻る
exp_error       endp
;
; -------------------- 一文字戻し
;
eungetc         proc
        dec     si                      ; 実際にはテキストポインタを戻しているだけ
        ret
eungetc         endp
;
; -------------------- 一文字取り出し
;
; 入力レジスタ  SI = テキストポインタ
; 出力レジスタ  SI = 次の文字を指す
;               AL = 取り出した文字
;
egetc           proc
        mov     al,[si]
        cmp     al,0                    ; 終端子以降には進まないようになっている
        jz      _egetc0
        inc     si
_egetc0:
        ret
egetc           endp
;
; -------------------- 空白を無視する一文字取り出し
;
; 入力レジスタ  SI = テキストポインタ
; 出力レジスタ  SI = 取り出した文字の次の文字を指す
;               AL = 取り出した文字
;
egetc_skips     proc
        call    egetc
        cmp     al,' '
        jz      egetc_skips
        ret
egetc_skips     endp
;
; -------------------- 因子
;
facter          proc
        call    egetc_skips             ; 項の先頭が
        cmp     al,'('                  ; 括弧ならば
        jnz     _exp_NUMBER
        call    simple_expression       ; 単純式の評価をして
        call    egetc_skips
        cmp     al,')'                  ; 最後が閉じ括弧でなければ
        jz      _fact_RET
        jmp     exp_error               ; 式エラー処理へ

_exp_NUMBER:
        call    eungetc                 ; 読み戻して
        call    get_hvalue              ; 十六進数値を得る
        jnc     _fact_RET
        jmp     exp_error               ; エラーなら式エラー処理へ
_fact_RET:
        ret

facter          endp
;
; -------------------- 項評価（乗除算）
;
term            proc
        call    facter                  ; まず因子を評価し
        call    egetc_skips             ; 次の文字が
        cmp     al,'*'                  ; アスタリスクなら
        jz      _exp_MULTI              ; 乗算の処理へ
;
        cmp     al,'/'                  ; スラッシュなら
        jz      _exp_DIVIDE             ; 除算の処理へ
;
        call    eungetc                 ; それ以外は文字を戻して
        ret                             ; 終わり
;
_exp_MULTI:
        push    bx
        call    facter                  ; 次の項を評価して
        pop     ax
        mul     bx                      ; 乗算する
```

```
        mov     bx,ax
        ret

_exp_DIVIDE:
        push    bx
        call    facter                  ; 次の項を評価して
        pop     ax
        mov     dx,0
        div     bx                      ; 除算する
        mov     bx,ax
        ret

term            endp
;
; -------------------- 単純式評価の実体（加減算）
;
simple_expr_sub         proc
        call    term                    ; まず項を評価し
        call    egetc_skips             ; 次の文字が
        cmp     al,'+'                  ; プラスならば
        jz      _exp_PLUS               ; 加算の処理へ
;
        cmp     al,'-'                  ; マイナスならば（これは単項ではない）
        jz      _exp_MINUS              ; 減算の処理へ
;
        call    eungetc                 ; それ以外は文字を戻して
        ret                             ; 終わり
;
_exp_PLUS:
        push    bx
        call    term                    ; 次の項を評価して
        pop     ax
        add     bx,ax                   ; 加算する
        ret

_exp_MINUS:
        push    bx
        call    term                    ; 次の項を評価して
        pop     ax
        sub     ax,bx                   ; 減算する
        mov     bx,ax
        ret

simple_expr_sub         endp
;
; -------------------- 単純式の評価
;
simple_expression       proc
        call    egetc_skips
        cmp     al,'-'                  ; 先頭が単項のマイナスなら
        jnz     _sel
;
        call    simple_expr_sub         ; 単純式を評価した結果を
        neg     bx                      ; 負の値にする
        ret
;
_sel:
        call    eungetc                 ; 読み込んだ文字を戻して
        jmp     simple_expr_sub         ; 単純式の評価をする
;
simple_expression       endp
;
; -------------------- メインルーチン
;
main            proc
        mov     si,PSP_PARAM            ; DSはPSPを指している
        mov     ax,DGROUP
        mov     es,ax
        mov     di,offset param
        mov     cl,[si]
        mov     ch,0
        inc     si
rep     movsb
        mov     ax,es
        mov     ds,ax
        mov     byte ptr [di],NULL      ; ＡＳＣＩＺ文字列にする
        mov     si,offset param         ; これがテキストポインタ
        mov     ax,0                    ; 結果を返すレジスタ
;
        mov     initial_sp,sp           ; エラーで戻るときのための準備
;
        call    simple_expression       ; 式を評価して
        mov     ax,bx
;
        mov     di,offset dump_buffer
        call    conv_hword              ; 十六進文字列に変換し
        mov     byte ptr [di],NULL
        call    flush_buffer            ; 出力する
;
        mov     al,0
exp_exit:
        mov     ah,DOS_TERMINATE
        int     21h
main    endp

_TEXT   ENDS

_STACK  SEGMENT STACK
;
                dw      1024            dup (?)
;
_STACK  ENDS

_DATA   SEGMENT  WORD PUBLIC 'DATA'
;
param           db      130             dup (?)
;
dump_buffer     db      80              dup (?)
;
initial_sp      dw      (?)

_DATA   ENDS

        END     main

```

今月のお勧め

ご覧の通り単純なプログラムなので，今月はお勧めできるものがない．漫画に出てくるような気むずかしい料理人ならさっさと不貞寝してしまうところだろうが，筆者にはそういうわがままは許されていない．この記事のこの部分を読みたいために今月も980円払ったという人もいるだろう（筆者独自の情報網により，現時点で3名確認されている(嘘))，まことに申しわけない．ということで今月の反省に入る．

今月の反省

「MS-DOSツール・ラボ」というタイトルには，今月のプログラムは少々合わないかもしれない．ツールというよりは，サンプルプログラムというところだろう．しかし，今回の再帰とか式の評価などの基本的で，かつ重要なテーマは雑誌ではあまり扱っていることがなく，かといって専門書ではひょっとしたら意地悪しているんじゃないかと思うくらいにむずかしい書き方をしているものが多いので，いつか書きたいと思っていた．「おもしろかったよ」とひとこと感想を寄せていただくと，「スジが違う」と編集の方に怒られなくてすむ．

次はスタックの話でもしたいと思っているのだが，2回続けて「この手」では編集部から怒りの電子メールが飛んでくる恐れがある．来月はなにか少しは役に立つツールをやらねばなるまい．

ということで，来月は今月のテーマの再帰を使って，指定されたファイルを検索して表示するプログラムを作ることにする．大容量のハードディスクをおもちの方などは「え～と，あのファイルはどこにあったかな」と探しまわったことがあるだろう．そんなときのために，指定されたファイルをルートディレクトリからはじめて全部のサブディレクトリから再帰的に検索してくれるツールだ．再帰自体の説明は今月してしまったので，来月分はリストとプログラムの説明にスペースを十分割くことができそうだ．MS-DOSのメモリ関係のファンクションにも触れることになるだろう．

参考文献：

●LISP　やさしい記号計算入門　ダビッド・S・トーレッキー著　啓学出版
●ソフトウェア考現学　荻谷昌己著　CQ出版社

98活用アセンブラ技法

ビデオシステム編
第6回
バックカラー付き画面表示ルーチンの作成

星野 操

前回作成した1文字読み書きルーチンに続いて，今回は複数文字を一括して読み書きするルーチンを作成します．これによって，よりオーバーヘッドの少ないバックカラー付き画面表示が可能になります．

複数文字を一括して読み書きする

オーバーヘッドを減らし，処理速度を向上させる

さて，今回は前回に続いて，バックカラーをサポートした表示ルーチンを作成していきましょう．前回制作したルーチンは，1文字単位の画面の読み書きをサポートするものでした．しかし，前回コメントしたように画面のすべてのアクセスをこれらのルーチンを利用して行うのは，現実的ではありません．あまりにオーバーヘッドが大きくなってしまうからです．

今回は，複数の文字を一括して読み書きするルーチンを用意することにします．これによって，同一の量の読み書きをする場合の手間が軽減され，高速になることが期待されます．呼び出し速度が改善されることに加えて，「連続した」アクセスに伴う各種演算の簡略化が期待されるからです．

たとえば，前回の例のようにバックカラーを設定するためには，グラフィック画面でのアドレスを計算する必要があります．このためには，通常除算と乗算（シフト演算だけでも可能ですが）が必要になります．乗除算は比較的時間がかかる演算ですから，1文字表示するごとに，このような演算をするとかなり遅くなってしまいます．

しかし，連続した文字を書くことがわかっていれば，グラフィック画面のアドレスは，単にインクリメントをするだけで計算できるのです．そして，画面右端であるときのみ補正を行う（80×15をたす）だけですむのです．

このように，連続した文字を一括して書き込めるようにすることは，表示速度を改善するうえで大きな意味があるわけです．

用意するサービス

では，実際にどのようなサービスを用意するかです．考えられるサービスについては前回リストアップしてありますが，今回は以下のようなものを用意することにしました．また，いずれのサービスについても転送をワード形式（文字コードはPC-9801の画面コードを直接使用し，アトリビュートについても2バイトを使用することで全文字・全アトリビュートを表示可能にする形式）とバイト形式（コードはSJIS，アトリビュートはバイトサイズにエンコードされた形式で，プログラム中の文字列を書き込むのに便利な様式）の双方を用意することにしました．

用意した内容としては，

①バッファから画面への書き込み
②バッファへの画面からの読み込み
③画面内部での移動
④画面へのコードの埋め込み
⑤バッファからアトリビュートデータの転送
⑥バッファから文字列の転送
⑦固定アトリビュートの埋め込み
⑧固定文字コードの埋め込み
⑨アトリビュート固定の文字列の書き込み

となっています．画面上でウィンドウを構成する場合など，すでに書かれている画面の情報を一括して保存し，それを復元するためには，①および②のサービスをワード形式で呼び出すのがよいでしょう．また，画面内部で縦横のスクロールをする場合には，③のサービスを利用するのがよいでしょう．③のサービスだけは，バイトとワードの区別がありません．

画面の一部クリアなどは，④のサービスを使用するのがてっとり早いでしょう．⑤⑥は，それぞれ他方の情報に変更をもたらさずに一方の情報を変更するものですが，使用頻度は比較的低いかもしれません．⑦⑧もそれほど使用されることはないでしょうが，バックカラーなどを一括して変更したい場合に使えるかもしれません．

⑨は非常に有用なルーチンで，プログラム中のSJISコードの文字列を表示する場合などに便利に利用できます．文字列中の特定のコードのアトリビュート（バックカラーなど）を変更したい場合は，さらに⑦のサービスを併用すればよいことになります．

複数文字を一括してアクセスできるようにしたのは，呼び出しの手間のほかに速度を重視したためです．前回の1文字読み書きルーチンは，比較的実行速度には甘く作ってありましたが，今回は多少気を配る必要があります．

引数渡し

まず呼び出しのときの引数は，1文字読み書きの場合と同じようにスタック渡しとします．スタック渡しとするのは，ひとつは高級言語から呼び出しやすいという理由からですが，実際問題として，ワークエリアを利用しないとプログラムを記述するのは困難ですから，スタック上に変数を取るのはしかたがないといえます．

絶対アドレスに変数を取る場合とスタック上に変数を取る場合とどちらが優れているかですが，案外スタック上の変数を利用したほうが効率が上がります．まず，オフセットが小さな場合(配列を引数やローカル変数に取るような場合以外）では，スタック上の変数を使用したほうが，絶対アドレスを利用するより1バイト小さくなります．これは速度の面でも有利です．

問題となるのはEA（実効アドレス）の計算ですが，286以上のCPUではアドレッシングモードによる実行クロック数の変化はほとんどないはずです．8086ではスタック上のアドレスを計算するのに時間がかかるので不利になるでしょうが，今回はあまり配慮する必要はないでしょう．

もう1つの利点としてDS(データセグメント）に依存せずに変数がアクセスできるということがあります．今回のような目的では，「呼び出し」側のプログラムのDSとともに，画面もアクセスする必要があります．さらには，内部的なワークエリアを参照する必要もあります．このような場合に変数がSS(スタックセグメント)でアクセスされるスタック上のものであれば，いちいちDSを復元せずに値を利用することができますから，明らかに有利です．

そのかわり，BP（ベースポインタ）を利用することができなくなりますし，一般にenter／leave系列の命令を関数の最初と最後で利用すると，サイズ・速度両面で多少不利になります．

しかしトータルで考えると，スタック上の変数の利用は高機能CPUでは推奨されるべきこと，と考えてかまわないでしょう．もちろん，レジスタを最大限に活用してからの話ですが．

漢字表示

バイト単位で文字コードの書き込みをする場合には，SJIS文字列の1バイト目と2バイト目をまたがないようにします．つまり，1バイト目を表示してからそれを中断し，別の場所に文字を書いてから2バイト目の続きを書くようなことは許さないということです．

たとえば，J-3100のBIOSを利用した日本語出力は同様な仕様をもっています．漢字表示は，中断せず連続して書き込まなければいけません．これに対して，AXなどハードウェアを利用したシステムでは，最終的に2バイト目が書き込まれれば，まったく問題なく漢字を表示することができます（ただし，1バイト目が書かれてから2バイト目が書かれるまでの間は文字が化けるはずですが)．DOS/Vなどのシステムはその折衷的位置づけで，書き込みのあとアップデートする範囲を適切に選ぶことによって，正しい漢字表示を行うことが可能です．

このような自由な書き込みを許容しようとすると，どうしてもSJIS表記の現在の画面コードを格納しておく場所を用意しなければなりません．そうしないと，たとえば画面に表示されているコードを一度JIS→SJIS変換して，それと新しく書き込まれるコードからSJIS→JIS変換を行い表示する，というような手間を踏む必要があります．

前回の1文字書き込みルーチンでもこの連続性を保つ必要がありましたが，今回も同様であり，かつ書き込み文字列の先頭と末尾もアラインメントが取られている必要があります．

前回とは異なり，今回は「SJISコードの2バイト目」を知ることができる(バッファの先読みをすればよい）ので，より効率的にコード生成をするこ

とができるのです.

マクロの使用

次にマクロをどれぐらい使用するか,いいかえればインラインの展開をどれだけ使用するかが問題になります.バイト単位の命令では,前記のようにSJIS→JIS変換が多用されますし,読み込みの場合にもJIS→SJIS変換が使用されます.とくに読み込みの場合には,JIS→SJIS変換の連続性は使用せずに文字ごとに変換を行っています.このため,この部分をマクロ化することは十分に意味があると思われます.実際にプログラム中で使用しているのは,前回と同様にサブルーチンをマクロで呼び出しているだけですが,この部分をマクロ展開することで高速化がはかれます.もちろん,プログラムサイズは大きくなります.

前回のJIS←→SJIS相互変換のサブルーチンは,いずれもレジスタ渡しだったので,それらの内容をそのままマクロで展開するようにすれば,簡単にマクロを使用するように変更できます.

同様に,アトリビュートのデコード・エンコードもマクロ展開することによって高速化がはかれるでしょう.

今回掲載するプログラムでもマクロ定義が行われていますが,これらは高速化のためというよりは,単に各サブルーチンで利用される部分を共通して括り出したというだけの話です.

上記の各サービスは実行内容が似通っていますから,似たようなルーチンが出てくるのは当然のことです.共通部をなるべく1つのコードにして目的ごとに細部で分岐するという手も取れますが,速度的には不利になります.

レジスタの最適使用

マクロ使用の点から,またレジスタの最適使用の観点から,これらのルーチンではある程度レジスタの扱いを共通にしておいたほうが便利です.今回は次のようにします.

まず,AX/DXは破壊性の一時レジスタとしてのみ使用します.たとえば,アトリビュートのエンコード・デコードやバックカラーの設定などはDXを破壊しますので,これにあわせてあるわけです.

BXは画面上のオフセットを指し示すものとします.画面コードはワード単位ですから,BXの最下位ビットはつねに0です.これはそのままで画面アクセスができるほか,セグメント値を選択すればアトリビュートのアクセスもできますし,2000Hのオフセットを付加することで画面コードと同一のセグメント値でもアトリビュートを操作することができます.また,バックカラー処理のために拡張アトリビュートを操作する場合には,右に1ビットシフトすることでオフセットアドレスの計算ができます.もちろん,あとで左に1ビットシフトして補正する必要がありますが.

CXレジスタは,ループ用のワークを格納するカウンタ,またはバイト形式の書き込みをする場合に生成されたJISコードを記憶するレジスタとして使用します.

SIは一般にバッファを利用する場合のオフセットアドレスとして利用し,DIは画面に書き込みをする場合のバックカラー表示用グラフィックアドレスとして利用します.

ESは主として画面(VRAM)アクセス用のセグメントとして利用し,DSはときに応じて必要なものに切り替えることにします.ただし,set_backの呼び出しによってESが破壊されることに注意する必要があります.

利用するローカル変数

このほかに,いくつかの決まったローカル変数を利用しています.拡張されたproc疑似命令を利用すると,その直後にローカル変数を定義することができます.

マクロを展開するのに分岐命令とそのラベルが必要な場合,マクロ記述内に確定ラベルを書き込むことはできません.むりに書き込むと,複数定義エラーになります.このため,macro疑似命令の直後にlocal疑似命令でmacro内部で利用するローカルラベルの宣言を行うことができるようになっています.これらは,アセンブル中に一意のシンボルに置き換えられるわけです.

同様に,proc疑似命令のあとにlocal疑似命令を使用することができるようになっています.この疑似命令の引数に取られたシンボルが要求するだけのローカル変数エリアが,proc最初のenter命令によって確保されます.領域は通常の高級言語のスタックフレームと同様に,BPより下位,SPより上位のアドレスに取られます.これらの領域は,当然のことながらret命令によって自動的に解放されます(leave命令がアセンブラによって自動的に生成される).

さらに,これらのlocal変数を定義した文字マクロが自動的に定義されます.たとえばlocal疑似命令でlocal変数colを定義した場合,

```
col     equ     <word ptr [bp-2]>
```

というような,equ文が実行されたのと同様な結果が生成されます.このため,ローカル変数は通常の変数と同様にレジスタとの値のやりとりが可能となります.このあたりは,引数が自動的に文字マクロとして定義されるのと

同様に考えてよいでしょう．ここでは，共通に使用されるlocal変数としてcol，d_seg，t_seg，a_segなどをあげておきましょう．colは現在の画面ポインタ（BX）がさす文字のカラム位置を示すもので，実際には（80－カラム）となっています．ただし，画面内移動のときのみ実際のカラム数になることがあります．これは，画面端でのDIの補正を行うために使用するものです．

d_segはライブラリのデータセグメント（_DATA）を格納するもので，t_seg／a_segはそれぞれ画面のテキストセグメント，アトリビュートセグメントを指定するものです．これらは定数といえるのですが，セグメントレジスタには直接定数を読み込ませることはできないので，変数から読み込ませるようにしているのです．これも，DXレジスタなどを経由してセグメントレジスタに値をセットするのとどちらがよいかはむずかしいところですが，少なくともサイズの面では有利です．効率はとんとんぐらいでしょうか．

分岐処理の回避

分岐の回避についてですが，まずワードとバイトの処理の分化を積極的に行っています．バイト処理は文字コード・アトリビュートの変換がつきものです．これらはワード処理の場合は不要ですから，同一のループでバイト・ワードの処理を行おうと思うと，これらの分岐が非常にひんぱんに用いられるようになってしまいます．しかも，どちらの処理をするかを示すレジスタは用意できないので効率が悪化します．

ですから，コードが大きくなることは承知のうえで，初期設定が終わったあとモードによって分岐させ，それぞれのループを独立に形成させています．これによって，ループ中でも（モードに対する）無駄な分岐を省くことができます．もっとも，比較的使用頻度が低いと思われるルーチンで，しかも，それぞれのループのほとんどの部分が共用できる場合には，ループ中で分岐している場合もあります．

このほかに，より効率的に実行するために，普段実行される可能性が低い部分を選択的に条件分岐で実行するという方法があります．たとえば，バックカラーは前回述べたように変更されない確率のほうが高いといえます．ですから，変更がない場合に実行シーケンスが乱されないようになっているとより効率的なわけです．単に@ifでバックカラーのセット部分をスキップさせる方法だと，常時条件ジャンプが実行されるので非効率的です．

ただし，このようなやり方だと明らかにプログラムの可読性が悪化します．そこで，このようなジャンプの最適化は，サンプルとして1か所だけで用いることにし，ほかの部分は@ifなどの通常の条件マクロを使用して処理することにします．

テーブルの採用

高速化手法の1つにテーブルの採用がありますが，今回はほとんど行っていません．どうしてもテーブル領域が大きくなってしまうことと，セグメント指定がわずらわしいからです．

たとえば，画面上のオフセットから，バックカラーを設定するのに必要なグラフィック画面を取得するアルゴリズムを考えてみましょう．これらは1：1に対応し，前者をXとすると，後者は（X/2　div　80）＊80＊16+（X/2　mod　80）となるわけです．80の除算が入ってきますが，80には「5」の因子が入っているので，シフト演算では計算することができません．

いちばん高速で効率的なコードを生成するためには，画面オフセットとグラフィックアドレスとの1：1のマッピングテーブルを作成しておくことです．このようにすれば，1命例でグラフィックアドレスを取得することがで

リスト1

```
ATTR_OFFSET     equ     (ATTR_SEG - TEXT_SEG) shl 4

;-----------------------------
; グラフィックアドレスを計算
; calc_ga: calculate graphic address
; argument:
;       adr:screen address
; return:
;       DI: graphic address
;       col: 80-column

calc_ga macro   adr
        mov     ax,adr
        shr     ax,1
        mov     dh,80
        div     dh
        mov     dl,ah
        mul     dh
        mov     dh,0
        add     ax,ax
        add     ax,ax
        add     ax,ax
        add     ax,ax
        add     ax,dx
        xchg    ax,di
        sub     dl,80
        neg     dl
        mov     col,dx
        endm


;-----------------------------
; バイト形式の呼び出しでSJIS変換をする
; expand_code: expand code for byte mode
```

きます．結果として，DIレジスタにグラフィックアドレスを保存しておく必要もなくなりますし，画面端での補正を行う必要もありません．その代償として，4KBにもなるマッピングテーブルとそれをアクセスするためのセグメントレジスタの値の設定が保証される必要があります．もっとも，後者はset_back_colのインプリメントしだいで問題がなくなります．

4KBのテーブルが無駄であると思われるのであれば，部分的なテーブルを利用することができます．たとえば，80は16で割り切れます．ですから16キャラクタごとのマッピングテーブルを用意し，その変換された値に16の剰余をたしてやることによって，アドレスを計算することができます．16でのわり算，および剰余は，シフト演算と論理積で得ることができますから，比較的高速に計算できます．テーブルの大きさもすべてをマッピングする方法の16分の1になります．ただし，この場合も変換の重さはさほど軽いとはいえないので，1文字書き込むごとにこの演算をするのは好ましくないと思われます．画面端の特殊処理をするだけのほうが効率的でしょう．

テーブルを使わなくても，最初のアルゴリズムに忠実にしたがって計算をすることはできます．ただし，乗算除算が必要なので遅くなることはしかたありません．もっとも，乗算は最適化することができるので，シフトと加算だけで実行できます．毎回この演算をするのは効率的ではないので，前記のように最初の1回だけ計算しておき，あとはインクリメントを繰り返し，画面端にきたときだけ補正するという方法を取ることにします．今回のプログラムは，このアルゴリズムを使用し，それをマクロ展開しています．

```
; argument:
;       CX: saved code
;       AL: code byte
;       SI: string pointer
;       si_offset: offset value to next code
; return:
;       AX: expanded code
;       CX: next saved code

expand_code     macro   si_offset
        or      cx,cx                           ;; check full/right side
        @if     nz
                ;; full width right side char
                xchg    ax,cx                   ;; get saved code
                mov     cx,0                    ;; clear work area
        @else
        mov     ah,al                           ;; check Sjis first byte
        ror     al,1
        add     al,0c0h
        not     al
        test    al,81h
        @e_if   z
                ;; full width left side char
                mov     al,ah
                mov     ah,[si+si_offset]       ;; sense forward
                sjis_to_jis <>
                sub     al,20h
                mov     cx,ax
                or      cx,8080h
        @else
                ;; half width char
                mov     al,ah
                mov     ah,0
        @endif
        endm


;----------------------------
; バックカラーの書き直しのチェックと変更
; check_back_color: check and update back color
; argument:
;       ds_value: ds reload value

check_back_color        macro   ds_value, es_value
        shr     bx,1
        ifnb    <ds_value>
                mov     ds,d_seg
        endif
        cmp     attr2nd[bx],al
        @if     nz
                ;; set back color
                mov     attr2nd[bx],al
                set_back <>
                ifnb    <es_value>
                        mov     es,es_value
                endif
        @endif
        shl     bx,1
        ifnb    <ds_value>
                mov     ds,ds_value
        endif
        endm


;----------------------------
; bx/di/colをアップデートする
; advance_pointer: update bx/di/col
; argument
;       BX: text pointer
;       DI: graphic pointer
;       col: column counter

advance_pointer macro
        inc     bx
        inc     bx
        inc     di
        dec     col
        @if     z
                mov     col,80
                add     di,80*15
        @endif
        endm


;----------------------------
; 繰り返し文字数のチェックと終了
; check cnt: shortcut if need

check_cnt       macro
        cld
        cmp     cnt,0
        @if     z
```

この計算に必要なコードも小さいわけではありません．似たようなルーチンがこのマクロを多用すると，コードサイズの面で不利になります．そのような場合は，サブルーチンでまとめてしまうと速度の面ではもっとも不利になりますがコードサイズ(データも含む)の面ではもっとも有利になります．

以上のように，サイズと速度とのトレードオフを考えることになります．今回のプログラムで速度的に問題があれば，テーブリングを利用することによって速度の改善がはかれます．

プログラムの解説

back2.asm

それでは，プログラム自体について解説することにしましょう．

このプログラムは，前回のプログラムの直後にアペンドされるようになっています．ですからインクルードファイルなどの指定は行われていません．

ATTR_OFFSET

最初のATTR_OFFSETは前記のキャラクタセグメントを利用してアトリビュートを操作する場合のリニアオフセットの値です．

calc_ga

次のcalc_gaは，前の部分で詳しく述べているバックカラー処理用のグラフィックアドレスを計算するマクロです．画面に書き込むサービスは，このマクロを使用しています．引数adrをまずAXに読み込み，DXを補助レジスタとしてアドレスの計算をします．最後にDIにアドレスを設定し，かつローカル変数colにコラム位置(画面端補正用)の値を設定します．

```
                        ;; shortcut
                        ret
                @endif
                endm


;-----------------------------
; データを画面に転送
; move_to_screen: move data into screen
; argument:
;               mode:1 for word, 0 for byte
;               adr:offset address on screen
;               cnt:characters to move
;               srcoff:source offset
;               srcseg:source segment
; return:
;               none
; broken:
;               all except si/di
; cld direction

func    move_to_screen  <mode, adr, cnt, srcoff, srcseg> <si di>
        local   col, d_seg, t_seg

        check_cnt <>

        ; setup local variables / registers
        calc_ga <adr>

        mov     d_seg, _DATA
        mov     t_seg, TEXT_SEG

        push    ds
        mov     ds, srcseg
        mov     si, srcoff
        mov     bx, adr

        mov     es, t_seg

        cmp     mode, 0
        @if     z
                mov     cx, 0                   ; code saver
                @begin
                        ; get code into ax
                        lodsb                   ; get character code
                        expand_code <1>         ; expand code

                        ; store it
                        mov     es:[bx], ax

                        ; get attr into ax
                        lodsb
                        decode_attr <>

                        ; save it
                        mov     es:[bx+ATTR_OFFSET], al

                        ; check back color
                        mov     al, ah
                        check_back_color <srcseg>, <t_seg>

                        advance_pointer <>

                        ; loop
                        dec     cnt
                @until  z
        @else
                mov     cx, cnt                 ; loop counter
                @begin
                        ; get code
                        lodsw
                        mov     es:[bx], ax

                        ; get attr
                        lodsw
                        mov     es:[bx+ATTR_OFFSET], al

                        ; check back color
                        mov     al, ah
                        check_back_color <srcseg>, <t_seg>

                        advance_pointer <>

                        dec     cx
                @until  z
        @endif

        pop     ds
        ret
endf

```

expand_code

次のexpand_codeは，バイト形式で呼び出しを行っているときに，SJIS変換をするものです．CXは前記のように，SJIS漢字の2バイト目を処理するためのものです．CXが非ゼロであれば，これは2バイト目処理ということで，ALに格納されている2バイト目のキャラクタは無視してCXの値をAXに設定します．CXは次の呼び出しに備えてクリアします．

そうでなければ，前回の1文字書き込みのルーチンと同様のアルゴリズムでSJIS 1バイト目の判定をし，それが満たされれば1文字「先読み」をしてからSJIS→JIS変換をします．そして画面表示コードに変換し，CXにそれを格納してフラグを設定します．次の呼び出しには，このCXに格納した値が利用されるのです．

si_offset

si_offsetは，先読みのときのオフセットで，バッファに文字コードのみが格納されているときには0，アトリビュートがマージされているときには1になります（SIは事前にインクリメントされているから）．いずれもが満たされない場合には，半角文字としてそのまま処理しています．

1文字書き込みのときと異なり，出力アドレスの連続性のチェックが不要になっていることに注意してください．

check_back_color

check_back_colorは，バックカラーの書き直しの必要性をチェックし，必要ならset_backを呼び出してバックカラーの変更をするものです．DSが_DATAを指していない場合は，ローカル変数d_segに設定された値を読み込

```
;-----------------------------
; 画面からデータを転送
; move_from_screen: move data from screen
; argument:
;       mode:1 for word, 0 for byte
;       adr:offset address on screen
;       cnt:characters to move
;       dstoff:distinateion offset
;       dstseg:distination segment
; return:
;       none
; broken:
;       all except si/di
; cld direction

func    move_from_screen        <mode, adr, cnt, dstoff, dstseg> <si di>
        local   d_seg, t_seg

        check_cnt <>

        ; setup local variables / registers
        mov     d_seg, _DATA
        mov     t_seg, TEXT_SEG

        push    ds
        mov     es, dstseg
        mov     di, dstoff
        mov     bx, adr
        mov     cx, cnt

        cmp     mode, 0
        @if     z
                @begin
                        ; get data into ax
                        mov     ds, t_seg
                        mov     ax, [bx]

                        ; check full/half width
                        or      ah, ah
                        @if     nz
                                ; full width
                                add     al, 20h
                                mov     dx, ax
                                and     ax, 7f7fh
                                jis_to_sjis <>
                                test    dx, 8080h
                                @if     nz
                                        ; right side
                                        mov     al, ah
                                @endif
                        @endif
                        stosb

                        ; get attribute
                        mov     al, [bx+ATTR_OFFSET]
                        shr     bx, 1
                        mov     ds, d_seg
                        mov     ah, attr2nd[bx]
                        shl     bx, 1

                        encode_attr <>
                        stosb

                        dec     cx
                @until  z
        @else
                @begin
                        ; get data
                        mov     ds, t_seg
                        mov     ax, [bx]
                        stosw

                        ; get attribute
                        mov     al, [bx+ATTR_OFFSET]
                        shr     bx, 1
                        mov     ds, d_seg
                        mov     ah, attr2nd[bx]
                        stosw

                        dec     cx
                @until  z
        @endif

        pop     ds
        ret
endf


;-----------------------------
; 画面内のデータ転送
; move_between_screen: move data between screen
; argument:
```

みます．これによって，データセグメントのattr2ndが参照できるようになります．アトリビュートが異なっていればそれを設定し，set_backを呼び出します．最後に，必要ならDSの値をもとの値に復元します．バッファを利用した文字列の書き込みなどの場合は，バッファのセグメント値に復元されるわけです．また，set_back呼び出しによってESの値が破壊されるので，引数es_valueの値を必要に応じてESに再設定します．

advance_pointer

advance_pointerは，画面ポインタBXとグラフィックアドレスポインタDIを更新するものです．BXは単に2を加えるだけですが，DIはインクリメントし，さらにcolをデクリメントした結果が0であれば80×15を加えます．これが，画面端での処理にあたります．colには，そのあと80を設定します．次の画面端で再び同様な処理が行われることになります．

check_cnt

check_cntは繰り返し文字数をチェックし，これが0であれば何も処理をせずに終了させるためのマクロです．

あとは，以上のマクロを利用して，実際に読み書きのサービスをする関数定義となります．転送をする場合には，オーバーラッピングの問題が生じることがありますが，画面とバッファとのやり取りの場合はオーバーラッピングが生じる心配はほとんどありませんから，ここではCLD方向に転送することに限定しています．画面内の転送ではこの問題が生じることがありますが，それはサービス内部で適当に処理をすることにします．ですから，転送をするバッファと画面のアドレスは，つね

```
;       srcoff:source offset address on screen
;       cnt:characters to move
;       dstoff:distination offset
; return:
;       none
; broken:
;       all except si/di
; overwrapping managed

func    move_between_screen <scroff, cnt, dstoff> <si di>
        local   col, step, step2, gstep, d_seg, t_seg

        check_cnt <>

        ; check direction
        mov     ax,srcoff
        cmp     ax,dstoff
        @if     nc
                mov     bx,cnt
                add     ax,bx
                add     ax,bx
                dec     ax
                dec     ax
                mov     srcoff,ax
                mov     ax,dstoff
                add     ax,bx
                add     ax,bx
                dec     ax
                dec     ax
                mov     dstoff,ax
                std
                mov     step,-1
                mov     step2,-2
                mov     gstep,-(80*15)
        @else
                mov     step,1
                mov     step2,2
                mov     gstep,80*15
        @endif

        ; setup local variables / registers
        calc_ga <adr>

        ; col correction
        cmp     step,1
        @if     nz
                ; std direction
                sub     col,80
                neg     col
        @endif

        mov     d_seg,_DATA
        mov     t_seg,TEXT_SEG
        mov     bx,dstoff
        mov     si,srcoff
        mov     cx,cnt

        push    ds

        @begin
                ; copy code/attribute
                mov     ds,t_seg
                mov     ax,[si]
                mov     [bx],ax
                mov     al,[si+ATTR_OFFSET]
                mov     [bx+ATTR_OFFSET],al

                ; check back color
                mov     ds,d_seg
                shr     si,1
                mov     al,attr2nd[si]
                shl     si,1
                check_back_color <>

                ; advance pointer
                lodsw
                add     bx,step2
                add     di,step
                dec     col
                @if     z
                        mov     col,80
                        add     di,gstep
                @endif

                dec     cx
        @until  z

        pop     ds
        cld
        ret
endf
```


ワンラインNEWS
ソードは32ビットCPUを搭載した高速デスクトップパソコンを東芝からのOEMで販売する．名称は「ソードJ-3100ZX171」で，初年度2,000台の販売を見込んでいる．

に先頭位置を示してやることにします．

move_to_screen

move_to_screenは文字データ・アトリビュートがマージされた形のバッファから画面への転送をするものです．引数には，バイト・ワード転送指定を示すmode，画面上のオフセットadr，転送文字数cnt，バッファのアドレスを示すsrcoff／srcsegです．さらにローカル変数として，前記のようにcolとd_seg／t_segを確保しています．

プログラムは，最初に転送文字数をチェックしたあとグラフィックアドレスを計算し，DS（d_segの値が入っていることが期待される）をpushしてからDS：SIにバッファのアドレスを設定します．同様に画面のオフセットをBXに設定します．

このサービスは，比較的ひんぱんに使用されることが期待されるので，1つのループの中でバイト・ワードの判別をせずに最初にモードによって分岐し，2つのループを作成しています．これによって，modeの判断は最初の1回ですみます．バイト転送の場合は，lodsbでコードを読み込んだあとexpand_codeで拡張し，それをテキストセグメントに書き込みます．続いて，同様にアトリビュートを拡張し，それをアトリビュートエリアに書き込みます．そしてバックカラーの部分について，check_back_colorで処理をします．最後に，advance_pointerで各ポインタを更新し，cntが0になるまでループを繰り返します．

ワードタイプの場合は，文字コードとアトリビュートの拡張が入らないだけで，処理の内容の流れは同じです．ただ，SJIS処理用にCXを使用する必要がないので，こちらは，高速化のためにCXをループカウンタに使ってい

```
;----------------------------
; コード・データを埋め込む
; set_to_screen: set code/attr into screen
; argument:
;         mode:1 for word, 0 for byte
;         adr:offset address on screen
;         cnt:characters to move
;         code:filling character
;         attr:filling attribute
; return:
;         none
; broken:
;         all except si/di
; cld direction

func    store_to_screen <mode, adr, cnt, code, attr> <si di>
        local   col, t_seg

        check_cnt <>

        ; check mode
        cmp     mode, 0
        @if     z
                ; expand attribute, code must be half width
                mov     ax, attr
                decode_attr <>
                mov     attr, ax
        @endif

        ; set graphic address
        calc_ga <adr>

        ; register set
        mov     t_seg, TEXT_SEG
        mov     bx, adr
        mov     si, code
        mov     cx, attr
        mov     es, t_seg

        ; loop
        @begin
                ; save data
                mov     es:[bx], si
                mov     es:[bx+ATTR_OFFSET], cl

                ; check back color
                mov     al, ch
                check_back_color <>, <t_seg>

                advance_pointer <>

                dec     cnt
        @until  z

        ret
endf


;----------------------------
; アトリビュートのみの転送
; move_attribute: move attribute only
; argument:
;         mode:1 for word, 0 for byte
;         adr:offset address on screen
;         cnt:characters to move
;         srcoff:source offset
;         srcseg:source segment
; return:
;         none
; broken:
;         all except si/di
; cld direction

func    move_attribute <mode, adr, cnt, scroff, scrseg> <si di>
        local   col, d_seg, a_seg

        check_cnt <>

        ; setup variables/registers
        calc_ga <adr>

        mov     d_seg, _DATA
        mov     a_seg, ATTR_SEG

        push    ds
        mov     ds, scrseg
        mov     si, scroff
        mov     bx, adr
        mov     cx, mode
        mov     es, a_seg

        @begin
```

ます．もちろん，ここで@untilを使用せずに，loop命令を使用できればそのほうが効率的ではあります．

最後に，DSを元の値に復元してから終了します．

move_from_screen

move_from_screenはその逆に，画面からバッファへの転送を行うものです．引数もほとんど同様で，バッファの意味がソースであるかディスティネーションであるかが異なるだけです．

モードによって別のループを組んでいるのも同様ですが，バックカラーをグラフィック画面に書く必要はないので，その計算処理などがなくなっています．レジスタの使い方は，ES：DIをバッファポインタとすることで，stos命令が有効に使えるようにしています．かわりに，DSを切り替えて画面などをアクセスするようになっています．CXは，ここでは両モードともにループカウンタとして使用しています．

ワード形式の場合は，そのまま画面コードおよびワード形式のアトリビュートを格納するだけであり，バイト形式はその前段階にコード・アトリビュートのエンコード処理が入ります．コードのエンコードは，ここではとくに気にせずJIS→SJIS変換を２回行っています．コードの同一性を判定することでこの変換を１回ですませることも不可能ではありません．また，そのアルゴリズムをJIS→SJIS変換ルーチンに埋め込むことも可能でしょう．

move_between_screen

move_between_screenは，画面内の転送です．このサービスは前記のように，オーバーラッピング処理をする必要があります．この場合，転送モードは意味をもたないので引数には取ら

```
                        ; get attribute
                        or      cx,cx
                        @if     z
                                ; half type
                                lodsb
                                decode_attr <>
                        @else
                                lodsw
                        @endif

                        ; set attribute
                        mov     es:[bx],al

                        ; check back color
                        mov     al,ah
                        check_back_color <srcseg>,<a_seg>

                        advance_pointer <>

                        dec     cnt
                @until  z

                pop     ds
                ret
endf


;----------------------------
; コードのみの転送
; move_string: move code only
; argument:
;       mode:1 for word, 0 for byte
;       adr:offset address on screen
;       cnt:characters to move
;       srcoff:source offset
;       srcseg:source segment
; return:
;       none
; broken:
;       all except si/di
; cld direction

func    move_string <mode, adr, cnt, scroff, scrseg> <si di>

        check_cnt <>

        ; setup variables/registers
        push    ds
        mov     ds,scrseg
        mov     si,scroff
        mov     di,adr
        mov     ax,TEXT_SEG
        mov     es,ax
        mov     bx,cnt

        cmp     mode,0
        @if     z
                mov     cx,0                    ; code saver
                @begin
                        ; get code into ax
                        lodsb                   ; get character code
                        expand_code <0>         ; expand

                        ; store it
                        stosw

                        ; loop
                        dec     bx
                @until  z
        @else
                mov     cx,bx
                rep     movsw
        @endif

        pop     ds
        ret
endf


;----------------------------
; 固定値の埋め込み（アトリビュート）
; set_attribute: set attribute only
; argument:
;       mode:1 for word, 0 for byte
;       adr:offset address on screen
;       cnt:characters to move
;       attr:attribute to set
; return:
;       none
; broken:
;       all except si/di
; cld direction
```


ワンラインNEWS
三洋電機はボーナス商戦に照準を合わせて，16ビットのノート型パソコンの国内販売を開始する．遅れをとっていたノート型パソコン市場の巻き返しをはかる．

れていません．オーバーラッピング処理のため転送方向は上位下位双方がありえますから，step／step2／gstepというローカル変数にステップ量（たとえば1または－1）を設定し，これをポインタレジスタに加算することでポインタの更新を行います．

まず最初にオーバーラッピングの可能性の判定をし，可能性がある場合にはポインタを末尾に変更します．step変数にも値を設定し，同時にディレクションフラグ・colローカル変数の値も操作します．あとは画面上のコード・アトリビュートを転送し，さらにattr2ndの値を参照して必要ならバックカラーを設定することを繰り返しているだけです．最後にCLDを実行してディレクションフラグを設定します．

set_to_screen

set_to_screenは，コード・データを埋め込むものです．バイト形式の場合は事前にコード・アトリビュートの拡張を行います．コードは半角文字でなければいけません．そして，データの埋め込みとバックカラーの設定を繰り返します．DSはデータセグメントを指したままであることが期待されるので，check_back_colorの引数は空になっています．

move_attribute

move_attributeは，バッファからアトリビュートのみの転送を行うものです．使用頻度は高くないと思われるので，ループを1つにしてアトリビュートのデコードはインラインで行っています．ただし，CXをモードにすることによって，毎回ローカル変数を参照することを避けています．

```

func    set_attribute <mode, adr, cnt, attr> <si di>
        local   col, a_seg

        check_cnt <>

        ; setup variables/registers
        calc_ga <adr>

        mov     a_seg, ATTR_SEG
        mov     bx, adr
        mov     ax, attr
        cmp     mode, 0
        @if     z
                decode_attr <>
        @endif
        xchg    ax, cx
        mov     es, a_seg

        @begin
                ; set attribute
                mov     es:[bx], cl

                ; check back color
                mov     al, ch
                check_back_color <>, <a_seg>

                advance_pointer <>

                dec     cnt
        @until  z

        ret
endf


;----------------------------
; 固定値の埋め込み（コード）
; set_string: set code
; argument:
;       mode:1 for word, 0 for byte (dummy)
;       adr:offset address on screen
;       cnt:characters to move
;       code:filling code (must half width char)
; return:
;       none
; broken:
;       all except si/di
; cld direction

func    set_string <mode, adr, cnt, code> <di>

        check_cnt <>

        ; setup variables/registers
        mov     di, adr
        mov     ax, TEXT_SEG
        mov     es, ax
        mov     cx, cnt
        mov     ax, code

        rep     stosw

        ret
endf


;----------------------------
; 固定アトリビュートで文字列を書き込む
; write_string: move code only
; argument:
;       mode:1 for word, 0 for byte
;       adr:offset address on screen
;       cnt:characters to move
;       srcoff:source offset
;       srcseg:source segment
;       attr:attribute to write
; return:
;       none
; broken:
;       all except si/di
; cld direction

func    write_string <mode, adr, cnt, scroff, scrseg, attr>
        local   col, d_seg, t_seg

        check_cnt <>

        ; setup local variables / registers
        calc_ga <adr>

        mov     d_seg, _DATA
```

move_string

move_stringは，文字コードのみの転送を行うものです．CXは全角文字コード記憶用に使用します．アトリビュートの操作が不要なので，転送はstoswなどだけで簡単に行われています．ワード形式の場合は，もっと簡単にrep movswが使用されます．

set_attribute/set_string

set_attribute／set_stringは，バッファとの転送ではなく固定値の埋め込みです．これらは，move_attribute／move_stringのバッファデータの読み込みがなくなったものと考えてさしつかえありません．

write_string

最後のwrite_stringが，もっとも一般的に使用されることが期待されるルーチンです．固定アトリビュートを指定して，バッファ内の文字列を書き込むものです．通常，SJISコード（バイトモード）であることが期待されます．バイトモードであれば，アトリビュートを拡張して引数であるローカル変数に格納しておきます．あとは，move_to_screenとほぼ同様な処理をしています．ただ，転送すべきアトリビュートをローカル変数から取り込んでいるところだけが異なっているといえます．

このルーチンは，サンプルとしてジャンプの最適化を行っています．check_back_colorとadvance_pointerの部分を展開していますが，マクロでは@ifで括られていた部分を別のルーチンに移し，必要な場合だけ制御を移すようにしています．こうすれば，多くの場合に制御構造の乱れがなくなることと実行速度が上がることが期待されます．ただし，可読性は悪くなります．

```
        mov     t_seg, TEXT_SEG

        push    ds
        mov     ds, srcseg
        mov     si, srcoff
        mov     bx, adr

        mov     es, t_seg

        cmp     mode, 0
        @if     z
                mov     ax, attr
                decode_attr <>
                mov     attr, ax

                mov     cx, 0                           ; code saver
                @begin
                        ; get code into ax
                        lodsb                           ; get character code
                        expand_code <0>                 ; expand it

                        ; store it
                        mov     es:[bx], ax

                        ; get and set attr into ax
                        mov     ax, attr
                        mov     es:[bx+ATTR_OFFSET], al

                        ; check back color
                        mov     al, ah
                        check_back_color <srcseg>, <t_seg>

                        advance_pointer <>

                        ; loop
                        dec     cnt
                @until  z
        @else
                mov     cx, cnt                         ; loop counter
                @begin
                        ; get code
                        lodsw
                        mov     es:[bx], ax

                        ; get attr
                        mov     ax, attr
                        mov     es:[bx+ATTR_OFFSET], al

                        ; check back color
                        mov     al, ah

                        ; expanded coeck_back_color <srcseg>, <t_seg>
                        shr     bx, 1
                        mov     ds, d_seg
                        cmp     attr2nd[bx], al
                        jnz     s2
                        s4:
                        shl     bx, 1
                        mov     ds, srcseg

                        ; expanded advance_pointer <>
                        inc     bx
                        inc     bx
                        inc     di
                        dec     col
                        jz      s3
                        s5:

                        dec     cx
                @until  z
                jmp     s1                      ; for special function
                        s2:                     ; contents of check_back_color
                                mov     attr2nd[bx], al
                                set_back<>
                                mov     es, t_seg
                                jmp     s4
                        s3:                     ; contents of advance_pointer
                                mov     col, 80
                                add     di, 80*15
                                jmp     s5
                s1:
        @endif

        pop     ds
        ret
endf
```

情報処理技術者試験
第2種合格！トレーニング
第3回　ソフトウェアの基礎Ⅰ

日本工学院専門学校　長澤洋一郎

ソフトウェアに関する出題の範囲と傾向

2種の試験で用いられている，ソフトウェアの基礎知識に関する出題の範囲は表に示した通りです．この出題範囲に基づき，平成2年度秋期までに出題された過去数回分の問題を解析し，出題の傾向をみてみます．

■ソフトウェア

①オペレーティングシステム，②プログラム言語，③ファイル編成，⑥アルゴリズムに関する出題が多くみられます．

①と②については，2回おきに出題される周期的な傾向がみられます．

③は「ファイルに関する基本的な考え方」と，「装置の容量計算，データ転送速度」にわかれて数多く出題されています．とくに，容量計算，転送速度に関する問題は毎回必ず出題されており，理解しておくことが大切です．

⑥も毎回のように出題されており，午後で出題される「流れ図」とあわせて勉強しておくとよいでしょう．

■情報処理システム

昭和62年度頃までの出題には周期的なものが見られ，出題頻度もかなり低いものでした．しかし，システム監査の試験が行われるようになった頃から，出題頻度は高くなり，この傾向はしばらく続きそうに思えます．

①システム処理形態，③システムの性能評価に関する出題が多くみられます．また，最近の傾向として，①と②，あるいは，③と②の2つの組合せで1つの問題を形成している場合があります．どちらか一方だけでなく，両方理解しておくことが必要です．

■その他

2種の試験の全体的な傾向として，1種で出題された問題の2種への転用があります．過去の1種の問題に目を通しておくとよいでしょう．

以上いくつかあげたポイントのうち，オペレーティングシステム，ファイル編成，プログラム言語に重点を置き，今回と次回で解説することにします．

ソフトウェアの基礎知識に関する出題範囲

1　ソフトウェア	
①オペレーティングシステム	オペレーティングシステム（制御プログラム）の目的，機能，役割，歴史など
②プログラム言語	アセンブラ，コンパイラ，インタプリタ，ジェネレータなどの言語の特徴，歴史，用語など
③ファイル編成	ファイルの定義，編成，アクセス方法に関する基本的な考え方および，各装置の容量計算，データ転送速度など
④ユーティリティ	各種ユーティリティプログラムの使用目的，使用方法，用語の意味など
⑤アプリケーション	各種アプリケーションプログラムの使用目的，使用方法，用語の意味など
⑥アルゴリズム	テーブルサーチ，マッチング，ソートなど一般によく使用されるアルゴリズムおよび，その図式化など
⑦データ構成・データフォーマット	ファイルの項目，レコード，ブロック，帳票設計，コード設計など
⑧プログラムの構成	プログラムのモジュール構成，モジュール結合に関する用語など
⑨プログラミング技術	プログラムを作るための効果的な手法など
⑩プログラムテスト	デバック，テストに関する手法，考え方など
⑪その他	データベースなど
2　情報処理システム	
①システム処理形態	オンライン，バッチ処理など
②システム構成	デュプレックス，マルチプロセッシングシステムなど
③システムの性能評価	応答時間，アクセス時間など
④その他	応用システムなど

オペレーティングシステム（OS）

OSの構成

OSは，「処理プログラム」と「制御プログラム」で構成されています．

■処理プログラム

ユーザーに直接サービスを与えるために用意されたプログラム群です．言語プロセッサ，アプリケーションプログラム，サービスプログラムなどがあります．

言語プロセッサとは，翻訳プログラムのことです．アセンブラやコンパイラなどがこれにあたります．

アプリケーションプログラムとは，特定の業務のために開発されたプログ

ラムのことです．表計算プログラムや図形作成プログラムなどがこれにあたります．

サービスプログラムとは，システムがユーザーのために用意した，システムのサービスを受けるためのプログラムのことです．リンカやローダなどがこれにあたります．

■制御プログラム

処理プログラムを管理するためのプログラム群です．そのため，制御プログラムだけをOSと定義する場合があります．ジョブ管理，タスク管理，データ管理などがあります．

OSの目的

■生産性の向上

OSの目的は「生産性の向上」です．これは，システム資源（制御プログラム，処理プログラムなどのソフトウェア資源，CPUやチャネルなどのハードウェア資源，プログラマやオペレータなどの人的資源）を効果的に活用することによって生産性を向上させる「システム資源の有効利用」と，計算機と利用者のインタフェースを充実することによって生産性を向上させる「使いやすさの追求」の2つの側面を持っています．

■資源の有効利用

システム資源の有効利用は，さらに，「スループット(処理能力)の向上」と「ターンアラウンドタイム(応答時間)の短縮」の2つの側面を持っています．

スループットとは，単位時間あたりの処理量のことです．

ターンアラウンドタイムとは，処理の要求から結果出力までにかかる経過時間のことです．

この2つの概念は，「ジョブの連続処理」と「マルチプログラミング」により実現されます．

■ジョブの連続処理

ジョブを実行するためには，プログラムのローディングや，入出力資源の割り当てと解放などの準備や後始末が必要です．

このような作業のうち，人手によるものをオペレーションといい，この間ジョブは実行できず，システムは待ち状態になります．

ジョブの連続処理は，「OSにオペレーションを代行させる」ことによって，「オペレーションによる，システムの待ち状態を排除する」ことを狙ったものです．

■マルチプログラミング

入出力命令は，入出力チャネルや入出力制御装置によって処理されるため，入出力命令が行われている間，CPUは使用されず空き状態になります．

マルチプログラミングは，「複数のジョブで1つのCPUを共有する」ことによって，「入出力命令実行による，CPUの空き時間を排除する」ことを狙ったものです．

OSの機能

制御プログラムを構成するジョブ管理，タスク管理，データ管理の3つについて解説します．

■ジョブ管理

ジョブとは，ユーザーがシステムに与える仕事の単位のことです．システムに与えた仕事が，1つのプログラムから構成されている場合は「プログラム＝ジョブ」となり，3つのプログラムから構成されている場合は「3プログラム＝ジョブ」となります．

ジョブ管理は，ジョブを実行するための準備や，実行後の後始末を行います．

- ●ジョブの入力を行います．
- ●ジョブが使用する資源（データセットなどのソフトウェア資源，CPU，主記憶装置などのハードウェア資源）の割り当てを行います．
- ●使用した資源を解放します．
- ●ジョブの出力を行います．
- ●オペレータからのコマンドの入力やシステムからのメッセージの出力など，オペレータとシステムのインタフェースの役割を果たします．

■タスク管理

タスクとは，OSにおいて仕事として認識される最小の単位のことです．OSは，タスクに対してCPUを割り当てます．

タスク管理は，ジョブ（タスク）の実行の制御を行います．

- ●タスクが中断した場合（マルチプログラミングにおけるタスクの切り替え，入出力命令の実行，プログラムエラー，ハードウェアエラーなど），その理由を調べ，適切な処理をする「割り込み制御」を行います．
- ●マルチプログラミング実現のために，割り込み制御を用いて，各タスクへのCPU割り当ての切り替えを行います．
- ●タスクの実行中に，動的に変化する資源(主記憶装置など)を監視します．
- ●中断したタスクを再開するために，タスクがそれまで使用していた各種レジスタの保存と復元を行います．
- ●中断や実行を繰り返すタスクの状態を制御します．

■データ管理

データセットを体系的に管理し，データセットへのアクセスの制御を行います．

- ●各種データセットの編成法を提供します．
- ●各データセットに合わせた，各種アクセス法を提供します．
- ●データセットの作成や削除，更新な

どのメインテナンスサービスを行います.

- データセットの排他制御や，アクセス法の決定など，データセットへのアクセスの準備を行います.
- バッファの解放など，データセットへのアクセスの後始末を行います.
- 膨大な数のファイルを体系的に整理し，管理しやすくするシステム（カタログシステム）を提供します.

EXAMPLE 1

オペレーティングシステムに関する次の記述の中から正しいものを3つ選べ.

①オペレーティングシステムは，スループットの向上を目的としたものではなく，アプリケーションプログラムの管理を目的として作成されたものである.

②オペレーティングシステムは，制御プログラムと処理プログラムに大きく分けることができるが，コンパイラは制御プログラムの1つである.

③オペレーティングシステムの制御プログラムは，ジョブ管理やファイル管理は行うが，ハードウェアの入出力割り込みの解析や処理は行わない.

④オペレーティングシステムの制御プログラムは，処理プログラムが円滑にかつ効率よく実行されるように，計算機システムの資源を管理する機能をもっている.

⑤仮想アドレス空間をサポートするオペレーティングシステムは，必ずタイムシェアリングの機能をもっている.

⑥仮想アドレス空間をサポートするオペレーティングシステムといえども，その計算機が実装している主記憶容量以上の大きさのプログラムを動作させることはできない.

⑦仮想アドレス空間をサポートするオペレーティングシステムは，その計算機が実装している主記憶容量以上の大きさのプログラムでも動作させることができる.

⑧複数個のプログラムを，単一の計算機システムにおいて割り込みなどの手法をもちいて，見かけ上同時に実行することを，多重（マルチ）プログラミングという.

⑨多重プログラミングとは，1つのプログラムを複数個の計算機で実行させることをいう.

昭和58年度　第2種　問2

解　説

①スループットの向上は，OSの目的の1つです.

②コンパイラは言語プロセッサなので，処理プログラムに含まれます.

③割り込み制御は，制御プログラムであるタスク管理の大切な機能です.

④正しいです.

⑤タイムシェアリングのためには，仮想アドレス空間は必要ですが，仮想アドレス空間のために，タイムシェアリングは必要ありません.

⑥仮想アドレス空間とは，主記憶容量の大きさを意識せずにすむように，大きいメモリ空間を提供するためのものです.

⑦正しいです.

⑧正しいです.

⑨多重プログラミングとは，複数個のプログラムを1つのCPUで実行させることを言います.

解　答

④，⑦，⑧

EXAMPLE 2

オペレーティングシステムに関する次の記述中の［　］に入れるべき適当な字句を解答群の中から選べ.

オペレーティングシステムの代表的な機能には，ジョブ管理，タスク管理，データ管理などがある.

ジョブ管理は，利用者から処理依頼されたジョブを効率よく実行するために［　a　］の割り当てを行う機能である．ジョブ実行に必要な［　a　］とは，［　b　］，主記憶，ファイル，入出力装置であり，これらを明確に定義するために［　c　］が用いられる.

ジョブ管理によってジョブに与えられる［　a　］は，実行中に動的に変化する．これを管理するのがタスク管理である．仮想記憶方式の場合の主記憶の動的割り当て，マルチプログラミングにおける［　b　］の割り当て，［　d　］などがこれに該当する.

データ管理は磁気ディスクへのファイルのアロケーション，ディレクトリの管理などを行う機能である．また順次［　e　］，直接［　e　］，などの各種［　e　］法を提供する.

解答群

ア	CPU	イ	アクセス
ウ	課金情報	エ	コンパイラ言語
オ	資源	カ	ジョブ制御言語
キ	割り込み処理		

昭和62年度　春期　第2種　問8

解　説

a ジョブ管理のおもな仕事は，ジョブの実行の「準備と後始末」です．実行の準備とは，そのジョブが実行するうえで必要となる「資源」を，あらかじめ割り当てることです.

b ジョブに割り当てる資源には，入出力装置，入出力チャネル，主記憶装置，CPUなどのハードウェア資源と，処理プログラム，データセットなどのソフトウェア資源があります.

c OS(制御プログラム)に，ジョブの実行に必要なサービスの提供を要求するためには，専用の言語を用います.

この，OSに指示するための言語

を「ジョブ制御言語（Job Control Language:JCL)」といいます．

d ジョブ管理とともに，タスク管理もまた，ジョブの実行中に動的に変化する資源に限って，その監視をします．　動的に変化する資源としては，主記憶装置があります．

マルチプログラミングを実現するためには，一度に複数のジョブ（タスク）に対してCPUを割り当てなければなりません．そのためには，割り込み制御を用いたCPU割り当ての切り替えが必要です．

e データ管理は，順次編成や索引順次編成，直接編成などの各種ファイル編成法を提供するとともに，順次アクセスや直接アクセスなど，各々の編成法に合わせたアクセス法も提供します．

解　答

a　オ，　b　ア，　c　カ，
d　キ，　e　イ

ファイル編成

一般によく知られているファイル編成として，順次編成ファイル，索引順次編成ファイル，直接編成ファイル，区分編成ファイル，VSAMファイルの5つがあります．

このうち，2種の試験でよく出題されている，順次編成ファイル，索引順次編成ファイル，直接編成ファイルの3つについて解説します．ファイル編成法の違いからくる，各ファイルの構造やアクセス法などの特徴をつかんでください．

順次編成ファイル

バッチ処理システムでよく用いられる，もっとも基本的なファイル編成です．メインテナンス方法（ファイルの更新）に大きな特徴があります．

■ファイルの構造

ファイルの先頭より，連続してレコードが記録されています．

記録されているレコードは，これから述べるその他のファイル編成と違い，「キー」をもちません．

■レコードへのアクセス

レコードが「キー」をもたないため，順次アクセスしか行えません．

①　順次アクセス

順次アクセスとは，ファイルの先頭レコードから，物理的順番にしたがって，レコードをアクセスする方法です．

記録されている順番に，連続してレコードを処理する場合に向いています．

特定のレコードを処理する場合，あるいは，記録されている順番と関係なくレコードをアクセスする場合は，目的とするレコードまでの途中に存在する，不要なレコードもアクセスしなければなりません．

■ファイルのメインテナンス

レコードを更新，削除，追加して，ファイルを更新する処理のことを，ファイルのメインテナンスといいます．

順次編成ファイルは，メインテナンスに大きな特徴があります．

順編成ファイルのメインテナンスを考えるときには，ファイルが作成された媒体の特徴からくる，処理方法の違いに注意しなければなりません．

①　磁気テープ媒体

このファイルが磁気テープなどの，記録されているレコードが，アドレス（媒体上での記録位置）をもたないような媒体上に作成された場合は，レコードの更新，削除は，直接そのファイルに対して行うことはできません．

②　磁気ディスク媒体

磁気ディスクなどの，記録されている各レコードがアドレスをもつ媒体上にこのファイルが作成された場合は，削除と，書き換え後のレコード長が変わらないなら，更新が，それぞれ直接そのファイルに対して行えます．

しかし，①，②のいずれの場合においても，順次編成ファイルのレコードは連続して記録されているため，レコードの追加は，ファイルの最後に対しては行えても，任意の場所には行えません．

そのため，ファイルのメインテナンスは，新たに別の順次編成ファイルを作成し，そこへ更新したいファイルをもとにして必要なレコードを書き出す方法（ファイルの作り直し）をとります．

■その他の特徴

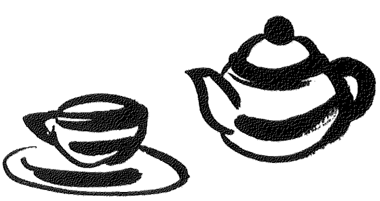
①記録されているレコードは，「キー」のような項目をもちません．また，ファイルの先頭より連続してレコードが記録されているため，各ファイル編成の中で，もっともスペース効率がいいです．

②順次アクセスだけが可能なので，このファイル編成だけが，すべての媒体に作成可能です．

③順次アクセスしかできないため，通常，このファイルを利用する処理プログラムの論理的順番（処理プログラムが処理したい，レコードの順番）にあわせて，物理的順番（レコードが記録されている順番）を変更します．

レコード中の氏名や従業員番号，伝票番号などの特定項目でソートします．

索引順次編成ファイルと，直接編成ファイルについては次回に解説します．



PRB

Programmer's Rest Bench

すき間ねらいで3回連続当選もでた創刊記念プレゼント余話

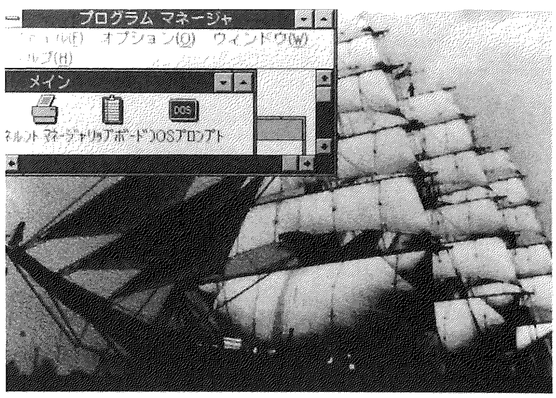
創刊号記念プレゼントでは，やはりパソコン，ハンディ98に人気が集中した．1月号から3月号までの各1名が激戦になった．そのほかでは，コンパイラ製品などに人気が集中した．好運にも当選された方がSE-NETに書き込んだメッセージは4月号でお伝えしたが，人気賞品をさけて応募し，3回連続プレゼントを獲得した読者もいた．ハンディ98やコンパイラなどの数百分の1の確立に賭けるよりも「実」をとったわけである．

プレゼントの発送はアルバイトの女性にお願いしたのだが，3回連続となるとさすがにめだってしまった．「あの，この人この前も，またその前も送っているんですけど，いいんでしょうか」．もちろん問題なし．公取委の関係で，同じ号で1人の人がだぶって当選してはまずいらしいが，月が違えばいいにきまっている．それでも，発送担当者は少ない確率に賭けたハズレはがきの山に気がとがめるらしく，「でも，やっぱりやめたほうがいいんじゃないですか？」といっていた．

付録ディスクにWindows 3.0の壁紙を入れました

お気づきになった読者も多いと思うが，5月号の付録フロッピーディスクから「.BMP」の拡張子をもつ画像データが入っている．前号のクイズランダムスクリーンの正解を，Windows 3.0のBMP形式に変換したものだ．5月号はKAMOME.BMPで，6月号はHANSEN.BMPだ．このファイルを作るまではひと騒ぎだ．写真は，クイズを出題する段階で決めなくてはならないから，バラバラにしてもあまりわかりにくくないオリジナル写真を探すか撮影する．イメージスキャナは，Macに接続されているので，そこで読み込んで色を調整する．GIFフォーマットにセーブしてWindows 3.0の画像データ変換プログラムに読み込む．そこで，さらにディザをかけたりして調整する．クイズ用のデータはRGBのベタにしてバラバラに，次号で使う正解のデータはBMP形式で保存する．「季節感のあるものを」「壁紙にしてきれいなものを」など，だんだん欲がでてくるから写真の選定がいちばん難しい．

壁紙の設定は，Windows 3.0の「制御パネル」を起動して，「デスクトップ」アイコンをダブルクリックして行う．壁紙のファイルは，あらかじめWindows 3.0のシステムが入っているディレクトリ（たとえば¥windows）にコピーしておく．Windows 3.0をおもちの方はぜひ試していただきたい．



CPU，メモリなどIC千個以上を収集したハードウェアのコレクター

5月号のアセンブラの特集で，各種CPUの写真を掲載した．協力してくれたK氏はICをはじめとするハードウェアのコレクターである．いまもっているのは，CPU周辺が約200個以上，メモリでは300個以上，デジタルICが500個以上，アナログは200個くらいとのこと．ほかにもAltair8800bとかIBMのハイブリッドICもあるとか．撮影当日は，そのうちCPUを中心にもってきてもらったのだが，「こっちがi4004，こっちが8080で…」と，続々とコレクションが出されるたびに，編集部でもどよめきがおこった．ちなみに，彼は飯田橋にある某大学の院生だが，定期券で秋葉原に行けるからという理由だけで総武線沿いに住んでいるのだそうだ．

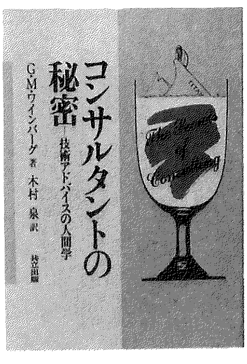
ワンラインNEWS
クボタコンピュータとミップス・コンピュータ・システムズは卓上型WS「RS 3330」を発売した．CPUには 33 MHzのRISC型チップ「R 3000 A」を使用している．

Book Review

■中村正三郎

■コンサルタントの秘密

G・M・ワインバーグ著
木村泉訳
共立出版　A5判　254ページ　2,400円



■評価チャート

うぶ	読者対象	すれっからし
1 2 3 4 5 6 7 8 9 10		

お手上げ	可読性	スラスラ
1 2 3 4 5 6 7 8 9 10		

なし	実用性	無限大
1 2 3 4 5 6 7 8 9 10		

季節もの	定番度	定番
1 2 3 4 5 6 7 8 9 10		

おじさん	ピョーキ度	ハッカー
1 2 3 4 5 6 7 8 9 10		

ゴールデンコンビによる「アドバイスの人間学」

著者のワインバーグ氏と訳者の木村氏のコンビは，ゴールデンコンビである．この二人の本であれば，まず手に取ってみることをお薦めする．期待が裏切られることはない．期待が裏切られたとしたら，その原因は何か考えてみよう．彼らが悪いのか，自分が悪いのか，はたまた社会か政府が悪いのか．こんなふうに考え始めた途端，あなたの期待は裏切られなかったことになる．あなたは，また新たな知見を得るからだ．

このコンビは，こんなふうに，ちょっと待てよ，別の観点から分析してみれば，といった発想のヒントを，観察力，洞察力を高めるアイデアを，組織，個人の意識を改革する話題を，ユーモアにくるんでさりげなく読者の心においてきた．『bit』誌の名物コラムであったイーグル村通信を連載していたのが，このコンビ．あれがなくなり寂しがったファンは多かった．当時，木村先生から頂戴したお手紙に「寂しいような，これで毎月締切に追われずほっとするような」とあった．以来，このコンビと毎月接する機会を我々は失ったわけだが，本書のような本の形でなら接することができる．そしてそれは，知的刺激に満ちた楽しい経験である．

「技術アドバイスの人間学」という副題がついている．「～の人間学」という題名は，このコンビの得意パターンだ．問題は結局，人間に行き着く（これは本書にある「コンサルタントの第二法則」）．ならば，よりよく人間を観察し，洞察する能力を高める必要がある．だから人間学．明快そのもの．

本書は，優秀なコンサルタントは，どういうふうに問題を捉らえ，分析し，適切なアドバイスを与えて解決しているか，そのマジックのタネ明かしをしている．といってよくあるハウツー本ではない．そんなバカなものをワインバーグ氏が書くわけがない．

マジックのタネは多くの法則としてまとめてある．これらは諧謔に満ちている．互いに矛盾するようなものもある．巻末には法則集の索引がある．様々なエピソードをユーモアを交えて紹介し，それから法則を導き出す．著者一流の得意パターンであり，本書の基本的骨格になっている．

知ってるだけでは，優秀なマジシャンにはなれない．使えなければならない．しかも適切な場面で適切なマジックを．これが難しい．本書でも，ヒントを繰り返し述べている．が，マジックをどう使うかというマジック，つまりメタマジックの秘密は，基本的には読者自身が考えることだとして，読者にゲタをあずけている．誠実な態度だ．評者には確たる答は得られなかった．確たる答えが得られたとしたら，それは間違っているに違いない．

法則Aはこんなときに使いなさい．でもAによって問題が解決できたと思った瞬間，新たな問題がはじまっており，ただちに法則Bを使わねばならない．だがそれは間違いで，ほんとはそもそもの最初からAではなくCを使うべきだったのだ．ただし，Cで問題が解決できたとしたら，ほんとのほんとはDを使った方がずっとよかったのだ．といった調子でまるで禅問答だ．

ふざけているのではない．ワインバーグ氏の話はいつもそうなのだ，常に足元を見直しつつも，大丈夫と思った瞬間，大丈夫でなくなる人間というも

のの危うさ，面白さを我々に語ってくれる．評価チャートは，全項目にわたって真黒にした．これもふざけているのではない．ダイナミックレンジが非常に広いのだ．例えば可読性は，訳もこなれており，読み流せばすいすい読めるが，考えながらだと一歩も先に進めない．実用性は全然ないけれど無限にある．といった具合．

本書でいうコンサルタントは，職業的コンサルタントだけではない．人から相談を受け，アドバイスする人を全て含んでいる．仲間同志の「あれ，どうしようか」「こうすればいいんじゃない」といった何気ない会話も，実際のところコンサルタントである．そう思って読めば，日常生活のいたるところで本書が応用できるだろう．評者は「技術アドバイスの人間学」から技術を取り払って考えたい．「アドバイスの人間学」である．アドバイスした方，された方の双方が，それによって成長していける．それこそよいアドバイスである．そのことを本書は教えてくれる．

本書を読み終えたら，同じコンビによる「システムづくりの人間学」，「ライト，ついてますか」，「計算機入力の人間学」（いずれも共立出版，一部共著，共訳）などに手を広げてみてはいかがだろう．

各シリーズ続々完結！注目はPen Windows関連書

古田島義和（神田・書泉グランデ）

91年4月号の『日経バイト』に，「ペン入力用2大OSの姿」という記事がありました．ペンを使ってコンピュータを操作するOS，米国のGO社から「Pen Point」が，Microsoft社からも「Pen Windows」が発表されました．これらが今後，パソコンの世界に新旋風を巻き起こすことになるのでしょうか?ともあれ，その解説書「The Power of Pen Point」（Addison Wesley／以下AW）は，話題性からも必見の1冊となることは確かなようです．

AT&Tの「UNIX System V/386」シリーズ24冊，GUIシリーズ10冊，ABIとBCSが今年7月にすべて完結する予定です．その中から6月に4冊の訳本が，トッパンから出版されます．「ユーザ・リファレンス・マニュアル」「プログラマ・リファレンス・マニュアル」「システム管理者リファレンス・マニュアル」「ユーザ・ガイド」で，基礎的なソフトウェアを構成し，プログラマの使うコマンド，システム・コール・サブルーチン，ライブラリなどの解説とUNIXシステムの概要を網羅しています．

OSF側のOS解説書もようやく出版されます．Prentice Hallから「OSF/1」シリーズ全4巻（6月），「OSF/1 Command Reference」「OSF/1 System & Network Administrator's Reference」「OSF/1 Programmer's Reference」（5月），「OSF/1 User's Guide」（6月），「Design of the OSF/1 Operating System」（9月）などです．

オブジェクト指向の解説書では，「The C++ Programming Language」（AW／Stroustrup著），「C++ Primer」（AW／Lippman著）のそれぞれ第2版が出版されます．続いての注目書は，「Objective-C:Object-Oriented Programming Techniques」（Pinson著）です．

前回とりあげた「ResEdit Complete」（AW）は第1位の座を獲得．「ResEdit Reference Ver. 2. 1」（AW）も8位と，ここ当分は「ResEdit」が上位を占めるでしょう．米国でも第4位と健闘しています．

89年に技術評論社のPascal版「コンピュータアルゴリズム事典」がロングセラーになりましたが，今年は「C言語による最新アルゴリズム事典」を出版，発売以来，第1位を続けています．同時期に啓学出版からBASIC版の「コンピュータアルゴリズム全科」が出版されましたが，C言語中心の世界で苦戦しています．第2位の「SUNシステム管理」（アスキー）は，7千円という高価格でありながら，2週間半くらいで100冊以上売り上げました．

■和書&洋書ベスト10

（書泉グランデ調べ：1991年3月1日～3月31日）

	和書	出版社	洋書	出版社
1	C言語による最新アルゴリズム事典	技術評論社	ResEdit Complete	Addison Wesley
2	SUNシステム管理	アスキー	Compiler	Addison Wesley
3	パソコン見栄講座	翔泳社	Undocumented DOS	Addison Wesley
4	コンサルタントの秘密	共立出版	Simple Book	Prentice Hall
5	rootから／へのメッセージ	アスキー	Programming Windows 2/E	Microsoft Press
6	入門C言語　改訂新版	アスキー	Object-Oriented Analysis	Prentice Hall
7	プログラミング言語C　第2版	共立出版	UNIX Network Programming	Prentice Hall
8	VZを256倍使うための本	アスキー	RedEdit Reference Ver.2.1	Addison Wesley
9	山田祥平の戦うMS-DOS	アスキー	Mathematica for the Science	Addison Wesley
10	実習C言語　改訂新版	アスキー	System Performance Tuning	O'Reilly

佐々木哲のニューヨーク N.Y. システム通信

SYSTEM N Y NEWS

失われるデータ

このコンピュータの時代に，いろいろな記憶媒体が現れてはより高密度なものに置き換えられていきます．紙，磁気，光磁気などなど……いまデータの寿命が問題となりつつあります．アメリカ連邦政府でも，数十年にわたる国勢調査などの膨大なデータの10%以上が磁気の劣化，機械のリプレースによる互換性のなさなどで読めなくなっていると聞きます．容量の大きさから，より重要なデータが蓄えられるようになってきている光ディスクなどでも，まだ寿命は実証されていません．10年ももたないという話もあります．

この分では人間の記憶がいちばん長もちしそうですね．

XGAをどう見るか？

VGAは，1987年のPS/2とともに登場し，いまではVGA以下のものを買う理由はなくなるまでになりました．VGA自体は，サードパーティにより1024×768やそれ以上までの高解像度のSuper VGAまで拡張されています．あとはどこに進むか．IBMは，VGAとともに紹介された8514/A規格を捨て去り，昨年暮れにXGAを発表しました．

では，次はすぐにXGAが標準となるのでしょうか．いろいろとむずかしい点があります．まず，現在のXGA規格はすぐに陳腐化しそうなこと．解像度の点からXGAは，1024×768を8514/Aと同様のinterlace付きで実現しています．すでに市場は，non-interlaceでそれ以上の解像度に向かいつつあります．また，XGAはいまのところPS/2の最上位機種でしか使えません．もちろん，ATバス用のものはありません．何よりも，いまのVGA並に普及するためには安価な互換製品が必要です．いままでの経験から，新しい規格に対する互換品が登場するまで2年はかかっています．

しかも，Super VGA，TIの32010，34020CPUベースのグラフィックボードが広まっており，なによりIBMの市場に対する影響力が絶対とはいえなくなってきています．ドライバひとつで，柔軟にハードウェアに対応できるMS-Windowsなどの普及も，ひとつの標準の必要性を減らしています．

IBM自身もXGAを普及させる姿勢を示しています．たとえば，XGAチップセット，資料のサードパーティへの提供，ATバス版の開発などを表明しています．

SVGAより24倍速い！ ATIの8514/Ultra

では，ここ2，3年はどの高解像度ボードを使うべきでしょうか．安価なのはSuper VGA．Tseng ET4000などのチップセットを使った高速なものが多数でています．せいぜい300ドル．でも，しょせんダムなボード．1024×768ではちょっと遅いでしょう．

Hercules Graphic Stationなど，34010ベースでVGAものったボードもお買い得です．一般のソフトとの互換性はVGAがのっているので問題ありません．MS-Windows，OS/2などでは高速な描画速度が得られます．値段も実売価格で700ドル以下とかなりお手軽です．残念なことに，肝心のMS-Windows 3.0の構造が悪く，graphic co-processorの機能が十分に生かせないという面があります．

そこでとてもホットな製品をひとつ．このコラムでもたびたび取りあげたカナダATI社[*1]の8514/Ultraです．8514/Aのモードに加え，1024×768をnon-interlaceでもサポートします．すでに死んだ規格8514/A互換のボードですが，これが実に高速！ MS-Windowsを走らせてSuper VGAより24倍，34010ベースのものより5倍高速，34020とくらべても2倍速いということです．もちろんXGAとくらべても値段が3分の1で3から10倍高速．もっとも，XGAは純正の8514/Aよりも遅いのですが．なんでも，MS-Windows，OS/2，AutoCADなど代表的なグラフィックアプリケーションを解析し，最適化された機能のASICを開発したそうです．開発者によれば50MIPSのRISC CPUが4インストラクションを同時に実行しているのに匹敵するとか．

さらにCrystal Fontと名付けられた，スケーラブルなフォントをハードウェアで提供しています．疑似的に4倍の高解像度をもったものです．専用のドライバが必要ですが，MS-Windows用のものなどが提供され

ワンラインNEWS
日本電気は若い女性向けのノート型パソコン「PC-9801NV（B）・（P）」の限定販売をはじめた．色はブルーとピンクで操作説明ビデオなどを標準添付しているのが特徴

ます．また，ATMのフォントを変換するユーティリティまで提供されます．もちろん，純正の8514/Aとハードウェア互換，純正よりも高速なAPI Turbo AIが付いてきますので，8514/Aをサポートした多くのソフトで使えます．

製品の形態は2つ．1つが8514/Ultra，定価599ドル．これは純正の8514/Aと同じくほかのVGAボードと動作します．1枚のボードでMicro Channel，ATバス両方をサポート．もう1つはVGA付きの8514/Ultra＋VGA，699ドル．これはATバス専用です．

＊1:ATI Technologies Inc., 3761 Victoria Park Ave., Scarborough, Ontario M1W 3S2, phone (416) 756-0718, fax (416) 756-0720

導入が容易なNetware 2.2登場

日本ではやっとLANが大騒ぎになってきたようですね．代表的なNOS，Netwareの日本のパソコンへの移植とともにさまざまなハードウェア，ソフトウェアが徐々に紹介されてきています．たりないのはサポートする人材だけ？　さて，Netwareは日本に根付くでしょうか．

日本でのNetwareの販売，サポート状況をアメリカでのものとくらべるといろいろ疑問がでてきます．LANというのはハードウェアの新規導入時にいっしょに売れればいいものではありません．むしろ，ある程度台数が増えてきてから導入するものです．

日本では，Netware 386 3.1Jとして，数多くあるNetwareの中でも最上級のものだけが提供されています．米国では，286CPU用のNetware SFT，Advanced，入門用といえる4ユーザーまでの限られたELS Ⅰ，8ユーザーまでで，それなりの機能をもったELS Ⅱなどなど，いくつかのレベルに分かれていました．それぞれのレベルの間では複雑ながらアップグレイド可能です．Netware 386では，高性能に加えNLMのサポートなど高度な利用が可能です．それだけにとても高価なものでした．

以前のv2.15のNetwareでは，インストレーションに何時間もかかり非常に複雑であるという問題を抱えていました．Netware 386のほうがサポートが楽だったのです．この3月にNovell Inc.はNetware 2.2を発表し上記の製品ラインを一新しました．インストレーションは，Netware 386よりも楽になったということです．ユーザー数によって値段を分け，容易な導入を可能にしています．

本当にNetwareを日本で普及させるためには底辺を広める必要があると思います．ただでさえ日本語化により高価になってしまったNetware 3.1Jを，いきなり導入するのはまるでオフコンを買うようなものです．10ユーザー版を用意するくらいでは無理です．

参考までに米国での新しいNetwareのラインアップをあげておきます．どれだけNetwareが身近なものかおわかりいただけるでしょう．

	Netware 2.2	Netware 3.11
5ユーザーまで	895ドル	
10ユーザーまで	1,995ドル	
20ユーザーまで		3,495ドル
50ユーザーまで	3,495ドル	
100ユーザーまで	5,495ドル	6,995ドル
250ユーザーまで		12,495ドル

今月のなんだこりゃ？

IBM互換機の世界でページプリンタといえば，Hewlett-PackardのLaserJetシリーズです．これだけ広まっただけに，カートリッジの種類は実にさまざま．フォントだけにはおさまらず，多様なエミュレーションカートリッジが出回っていました．

なかでも近年人気があったのが，PostScriptのエミュレーションカートリッジ．本体のメモリを2MB程度にする必要がありますが，本体コントローラの8MHz 68000でPostScriptを安価に実現します．互換だけではなく，本家Adobeの純正PostScriptカートリッジも入手可能です．価格は300～500ドル程度．

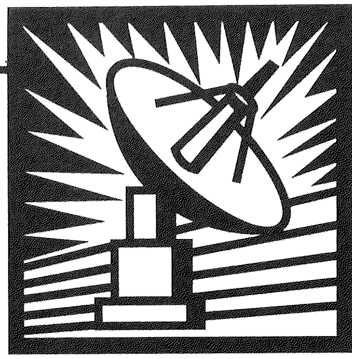
この方式だと，一般のPostScriptプリンタにくらべ，CPUの能力が劣るため処理速度が半分以下です．これにがまんできないパワーユーザーのために，カートリッジの代表メーカーPacific Data Products[*2]は，PacifcPage XLという新カートリッジを発売予定です．なんとこのカートリッジには，本体より強力なi960 RISC CPUがのっています．LaserJet本体の68000CPUと共同して動作し，そこらのPostScriptプリンタよりも高速に処理するとか．値段は未定．

この拡張性の高さにはただただ頭が下がる限りです．

＊2:Pacific Data Products, Inc., 9125 Rehco Road, SanDiego, CA 92121, phone (619) 597-4639, fax (619) 552-0889

SYSTEM N Y NEWS

Illustration : NOBUKI FUKUHARA



西川和久の
希望的観測

SYMDEB＝AUXなんてもう古い!!
ついに登場したWindows 3.0の第2世代開発環境

1 涙なくしては語れない!? いままでのWindowsデバッグ環境

最近，日本語Windows 3.0が各社から出荷され，これでやっと日本もWindows 3.0が立ち上がると業界は騒いでいるが，喜んでばかりはいられない．日本語Windowsアプリケーションが少ないのだ……．「一本でも多くの日本語Windowsアプリケーションを」と日夜開発に励んでいるエンジニアにとって，頭の痛い問題は開発環境である．

現在のSDK（Software Development Kit）に付属する環境では，なかなか効率は上がらない．「重すぎるCコンパイラ」「画面切り替えのCVW」「最後の切り札，SYMDEB＝AUX」などなど，涙なくしては語れない．本誌のWindows 3.0特集（1991年2月号）で，Windows-MakerやActorなどの開発環境を紹介していたが，しょせんプロトタイパーであり，最後にいきつくところはやはりSDKだ．しかし，今年に入ってからいくつかの本格的なWindows開発環境が米国で発売されだした．第2世代Windows開発環境の登場である．

2 MS-C 6.0＋SDKパックには意外なものが入っている

まずはじめに，今年の3月に発売開始となった「MS-C 6.0とSDKのセット版」である．個別で購入するより割安になっている以外一見まったく変わり映えしないものであるが，実は中にとんでもない代物が入っているのだ（私自身が実際に確認してはいないが，信頼できる関係者からの情報である）．それは，DPMI対応版MS-C 6.0と本当のWindowsアプリケーション版CVWである．

ご存知のようにMicrosoft Cは，コンパイル経過を中間ファイルに落とす古典的なコンパイラであり，このためコンパイル速度が遅い．また大きなコードを食わすと，コンパイラ自身のワークエリアがたりなくなり，処理が中断されてしまうこともよくある話だ．この2点を解決したのがDPMI対応版なのだ．DPMIのインターフェースを使い，中間コードやワークエリアを拡張メモリへ展開し，速度の向上や，より大きなコードを通すことが可能になった．

CVW自体は，DOSアプリケーションであり，画面切り替えでデバッグする環境である．使ってみるとすぐわかるが，画面が「バタバタ」スワップして，とても気持ちよく操作できる状態ではなかったのだ．しかし，これに含まれるCVWは完全なWindowsアプリケーションであり，Windowsアプリケーションのデバッグを Windowsの上から行える．いまから考えたら，あたりまえの環境がやっと登場したのだ（噂によると，次で紹介するBorland C＋＋にこそっと対抗したらしい……）．

ワンラインNEWS
ニフティは同社のパソコン通信「ニフティサーブ」でコンピュータウイルスが発見されたことを明らかにした．実害は報告されていないとのこと．

3 Borland C++ではWindowsのリモートデバッグができる

Turbo C++から早くも名前が変わりBorland C++となった，このコンパイラ最大の売りは，「SDKなしでWindows上で動くソフトが書ける」ということだ．これまで，Windowsに対応したコンパイラはいろいろあったが，リソースコンパイラやDialog Editorなどツール類が含まれておらず，SDKが必要だったため片手落ちであった．

このBorland C++，先のDPMI対応は当然行われているが（コンパイラだけでなく，TASM/TLINKまでもDPMIに対応している），「Whitewater Resource Toolkit」や「Microsoft純正のリソースコンパイラ」までが含まれている．デバッグ環境も強力である．

なんと，あのTDREMOTEのWindows版「WREMOTE」が含まれているのだ，開発環境とターゲットを分けてデバッグできる必殺のTurbo DebuggerでWindowsアプリケーションまでがデバッグ可能になった．古典的な技法，SYMDEB＝AUXなどとはくらべものにならないほど快適な環境がここに登場したのだ．もちろんTDWも加わったが，これは画面切り替え方式であるため，TD＋WREMOTEのほうがデバッグには圧倒的に有利である．

4 すべての開発環境がWindows上に Turbo Pascal for Windows

ここまではDOS環境＋Windows環境上で作動する，コンパイラやツール類であったが，Turbo Pascal for Windowsは「すべての開発環境がWindows上で作動」する．Turbo Pascal統合環境のWindows版と考えればイメージしやすいのではないだろうか．以前日経MIXで，「開発環境をすべてWindows上で」という話題でいろいろ論議があり，Windows自体のオーバーヘッドがあるため，「必要以上に重たくなる」という否定的な意見もあったが，これはこれでWindows開発環境の理想型だと思う．

今回は，Windows開発環境の最新情報を3つほど紹介した．米国でもでて間もないため，日本語版となるとちょっと時間がかかるであろう．しかし考えてみると，日本語Windows上でも米国版のWindowsアプリケーションは使えるのだ．DOS上で動く開発環境は，J-3100やAX，PS/55などを英語モードで使う必要はあるにせよ，工夫をすれば日本語Windowsアプリケーションを作るのに大きな手助けになるであろう．よけいなお節介といわれそうだが，開発環境については保守的な人も多い．しかし，いまどきSYMDEB＝AUXではあまりにも悲しいものがあると私は常々感じている．



コンピュータの法律

私的使用のためのプログラムのコピー

◆

熊谷信太郎（弁護士）

■複製が許される2つの場合

一般に著作者は，著作物を複製する権利を専有するとされているので，市販のプログラムを購入した場合に，そのコピーをとることは，原則的に禁止されています*1.

そして，例外的にコピーをとることが許されているのは，以下の2つの場合に限られます.

(1) バックアップ用のコピー*2
(2) 私的使用のための複製

このうち，バックアップ用のコピーに関する問題点は，前回までに説明しましたので，今回は，私的使用のための複製について，検討したいと思います.

■私的使用の範囲

私的使用のための複製とは，「個人的に，又は家庭内その他これに準ずる限られた範囲内において使用する」目的で複製する場合をいいます*3.

これにあたる場合には，著作物の複製が許されているのですが，しかし，「公衆の使用に供することを目的として設置されている自動複製機器を用いて複製する」場合は，私的使用のための複製にあたらないとされていますので注意が必要です．つまり，自分でコピーする場合に限って，私的使用のための複製と認められるのであって，業者にフロッピーの複写を依頼するような場合には，たとえ，個人的に使用する目的があっても，プログラムのコピーは許されないのです.

また，私的使用のためのコピーは，あくまで例外的にコピーが許されるのですから，個人的な使用であっても，大量にコピーすることはできませんし，家庭内での使用に限られますから，会社の業務に使用するためのコピーは許されません.

■バックアップ用と私的使用の違い

ところで，前回説明したバックアップ用のコピーと私的使用のためのコピーは，いずれも例外的にコピーが認められるケースとして著作権法に規定が置かれているわけですが，両者の違いとしては，まず，バックアップ用のコピーは，「プログラムの著作物の複製物の所有者」，すなわち，市販のプログラ

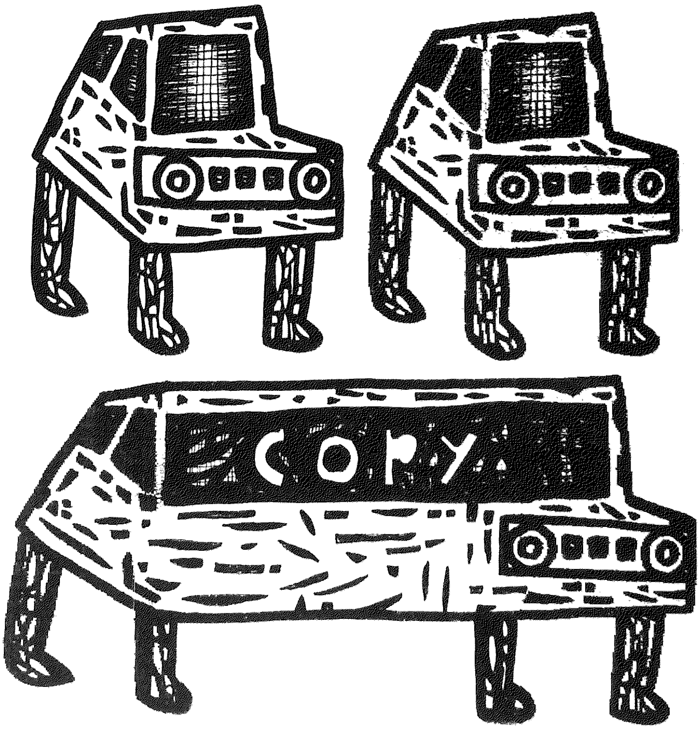


Illustration : Nobuki Fukuhara

ムを買ってきた人だけに認められた権利であるのに対し，私的使用のための複製は，市販のプログラムを買ってきた人だけでなく，他人のプログラムを借りた人も行うことができる点です．

また，バックアップ用のコピーは，プログラムの著作物についてだけ認められる例外ですが，私的使用のためのコピーは，プログラムの著作物だけでなく，レコード，出版物など，ほかの一般の著作物にも認められる点が異なります．

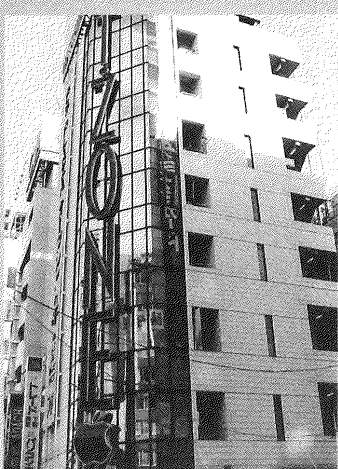
*1　著作権法第21条
*2　著作権法第47条の２第１項
*3　著作権法第30条

アキバ短信

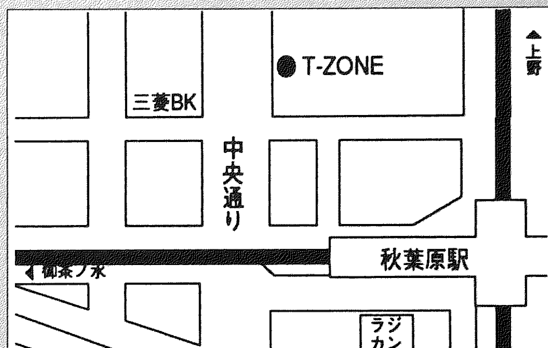
T-ZONE
アメリカンPCの牙城

本誌アンケートによると，プログラマーズ・ページの読者が使用しているパソコンで最も多い機種はNECのPC-9800シリーズだそうだ．世にパソコンは数々あれど，パソコンショップで売っているコンパイラやビジネスソフトも圧倒的多数はPC-9800シリーズ専用．互換機のエプソンも含めると，ビジネスパソコンのシェアの大半を占めることは想像に難くない．だが，近ごろその寡占状態にちょっとした変化の兆しが見られる．共通プラットフォームとしてのWindowsの登場，東芝のDynaBookシリーズなどのノートパソコン，「小学生になろう」というコピーでこの春大々的な宣伝を打ったMacintoshも確実にパソコンユーザーの裾野を広げつつある．

秋葉原でもPC-9800シリーズをメインにする店が多い中，T-ZONEはMacやIBM（とその互換機）の売り場の方が広いというめずらしい店だ．以前からMacのソフト・ハードを取り扱う数少ない店のひとつとしてMacユーザーの間ではよく知られていた（秋葉原へ行ったらイケショップとT-ZONEを回るのがMacユーザーのお定まりのコース）のだが，改装してますますMacをはじめとする外国PC寄りの売り場構成になった．電話機やパーソナルワープロが置かれていた１階が，Macintoshのハード売り場にリニューアル．以前から「アップルゾーン」の名で親しまれてきた地下１階がMacのソフト中心となった．



営業時間　10：00〜19：00　無休
TEL 03-3257-2650



２階は唯一NECや富士通，シャープなどの国内メーカーのパソコンとソフトウェア売り場．3，4階は「IBMゾーン」．４階にはIBM PC用の各種輸入ソフトウェア，３階は日本IBMの55Zや米国のIBM PC互換機が並ぶ．極東の島国を除けば世界中どこでもPCといえばIBM PCのことだという話は，プログラマなら一度は耳にしたことがあるだろう．漢字が表示できないという理由で日本ではまったく問題にされなかったIBM互換機も，ソフトウェアだけで日本語表示が可能なDOS（DOS/V）が発表されたことで，今後はビジネスパソコンの台風の目となる可能性は十分にある．ここでは外国人ビジネスマンが熱心に説明をもとめる姿もよく見られる．また，ゲームマシンの機能（セガのメガドライブ）とIBM互換機の両方の機能をもつことで話題になったテラも展示されている．２台のモニタをつないで，一方ではゲーム，他方ではWindows3.0が動作するというデモンストレーションは見るだけでも話のタネになるだろう．

5，6階は無線機，７階の書籍売り場はパソコン関連の雑誌のバックナンバー，洋書も揃っている．当然のことながらMacやIBM関連の書籍，雑誌も種類が豊富である．

●プログラマの仕事場　3

Programmer's Working Space

株式会社エム・ピー・テクノロジー

Windows 3.0をベースにマルチメディア時代を先取りする

文字，音，絵などの情報を統合して人に理解しやすいかたちで提示するハイパーテキストシステム．パソコン上でGUIを実現する新たなプラットフォームとして登場したWindows上で，ハイパーテキストシステムを実現するソフトウェア「日本語Guide」を販売し，同時にGuideの各種アプリケーションシステムを開発しているエム・ピー・テクノロジーは，システムよりのツールとマルチメディアアプリケーションの双方を開発しているユニークな会社だ．

開発の中心はWindows 3.0 プログラマの数がたりない

エム・ピー・テクノロジーの現在の仕事は，Windows関連が大半を占める．同社は，Windows 2.x用の「日本語Guide」を発売して以来，Windowsに強いソフトハウスとして知られてきた．Windows対応製品は日本語Guideだけではない．シャープのノート型パソコンAll in noteに標準でついてくるWindows用のアクセサリツールの開発，各種のWindows用デバイスドライバの開発も行っている．また，パッケージソフト「Windows SALAD」もシリーズ化して，トマト，パセリのように野菜に関連したネーミングのソフトウェアをだしていく予定だという．

現在，Windowsアプリケーションの開発ができるソフトウェア会社はまだそれほど多くはない．いきおい，実績のあるソフトウェア会社には，仕事も集まる．受託業務も今年はWindows 3.0関連のものがほとんどだと，同社取締役の若月氏は語る．Windows 3.0対応の自社製品を開発したりバージョンアップする作業と，受注しているWindows 3.0用のデバイスドライバやアプリケーションの開発をともに行わなければならない状態だ．

Windows関係の担当者は社内に数名，そのほかに外部から何名かの応援を頼んでいるが，Windowsのプログラマは，まだまだ数がたりない状態だという．「かえって，アメリカに頼んだほうが早いことも多いですね．アメリカではもうWindowsがほとんどですから」と若月氏．

低価格LANシステムをグループウェアのプラットフォームに

もうひとつ，力を入れているのがコミュニケーション関係のソフトウェアである．UNIXワークステーションをホストに，PC-9801シリーズを端末として，JUST-PCアダプタを使って電子メールや電子掲示板を利用するJust-linkや，G3準拠ファクシミリをパソコン上でエミュレーションするソフトウェアなども開発，現在OEMで販売している．

これに加えて，今年は新たにパソコン用のLANシステムを発売する予定だ．「Line-up」と名付けられたこのシステムは，パソコンのシリアルポートに専用

ワンラインNEWS
富士通はRISC型MPUを搭載したフルカラー3次元グラフィックアクセラレータ「FMR-3DGA」を発売した．対応機種は同社の「FMR-70」と「同80」．

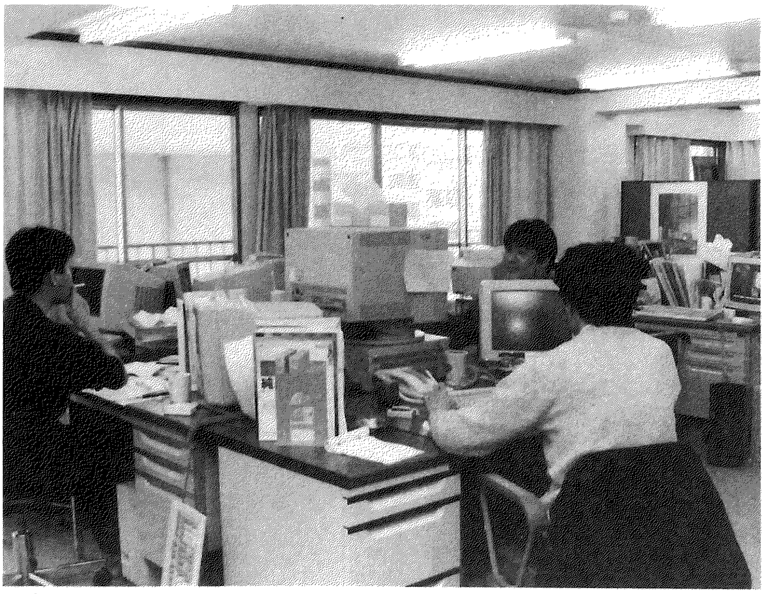
のアダプタを接続するだけで，簡単に構築でき，PC-9801シリーズの上位の機種を使えば，最大で19,200bpsの転送速度を実現できるという．しかも，このシステムをパソコン5台分のセットで10万円以下の価格で販売する予定だというから，小規模のLANシステムとして期待がもてそうだ．こういった低価格のLANシステムが普及することで，グループウェアの需要が高まれば，それをまたターゲットにして，Windows対応などのビジネスチャンスが増えていくことに期待しているのだという．

Guideアプリケーションの開発にはセンスが要求される

エム・ピー・テクノロジーには3つの部門がある．営業，開発，それにクリエイティブだ．ソフトウェア会社におけるクリエイティブとは何か？ ここでの仕事は日本語Guideを使ったアプリケーションの開発なのである．最近では，輸入商社のヤナセ向けレーザーディスクの画像データとCD-ROMの音声データをGuideで制御するマルチメディアプレゼンテーションツールを作成した．

Guideのアプリケーションのようなハイパーテキスト的なものを作成するには，いくつか方法がある．紙に絵コンテのようなものを書いて，線を引きながらその結びつきを考えていくこともあれば，シナリオライターに1本のストーリーを考えてもらって，ひと通りできたところで，ハイパーテキスト用に変更をくわえたりすることもある．文字，画像，音など，さまざまなメディアを扱うことになり，その見せ方，つまりストーリーや内容が重要性をもってくる．そこには，従来のソフトウェア開発が追求した，機能や処理速度とは違う問題がある．出版や映画・ビデオ制作に近い世界なのだ．クリエイティブという名称にしたのもそうした事情を反映している．

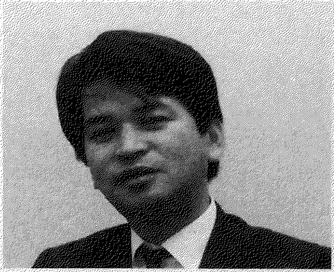
クリエイティブ部門のスタッフは，コンピュータやプログラミングとは縁のない職業――工業デザイナーとしてハードウェアのデザインをしていた人，グラフィックデザインをしていた人など，デザイ



開発部門の仕事場

About Company

吉本満寿夫社長，1986年設立．かつてデジタルリサーチジャパンの社員であった吉本氏は，将来はグラフィカルユーザーインターフェースの時代がくると確信して，MS-Windowsに取り組む．Windows用のソフトウェアを設計し，ハードウェアメーカーのOEM契約を勝ち取った．現在では，Windows用のアプリケーションソフトウェアの開発，販売にとどまらずWindows用のハイパーテキストシステム「日本語Guide」の販売と，その上で動くアプリケーションシステムの開発，受託業務，マルチメディア電子出版用のベーシックソフトウェアの開発，販売，電子メールシステムやファクシミリエミュレーションソフトウェア，低価格のLANシステムなど，コミュニケーションツールの開発，販売を行っている．また，1990年にはアメリカのメリーランド州に現地法人MPテクノロジーUSAを設立．



Location

JR中央線，地下鉄丸の内線四ツ谷駅から徒歩5分．周辺は新しいビルと古くからある建物が混在する雑然とした雰囲気がただよう町並み．

Products

「Win3 pro-Guide」(予価98,000円)
「Win3 Guide」(予価48,000円)

文書(文字)，グラフィックス，動画，音声などのさまざまな形態の情報を整理，体系化して表現するハイパーテキストを構築するためのソフトウェア．OWL/International社の「Guide」の日本語版．使い方次第で，アウトラインプロセッサ的な文書作成支援システムにもなれば，画像や映像を取り込んだマルチメディアデータベースシ

ステムにも，またCD-ROMやビデオ，LDのコントローラーとしても機能する（ただしビデオプロセッサ・ボード／CD-ROMなどのコントロールモジュールは，機種別対応で別売）．

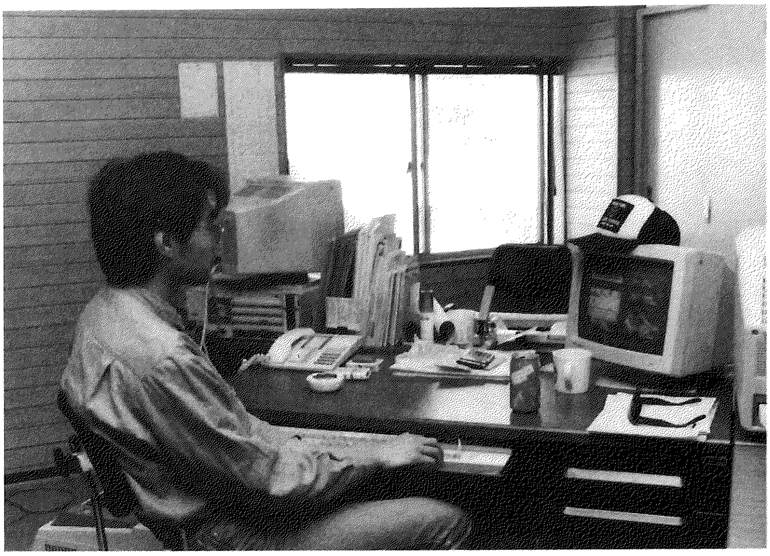
日本語GuideはMS-Windows上でのハイパーテキスト構築ツールである．MS-Windows 3.0に対応した最新バージョンは，まもなく発売予定．「WIN3 pro-Guide」では，ほかのMS-Windows対応アプリケーションをコントロール（DDE機能を利用）できる高級言語「LOGiiX」を搭載している．NEC PC-9801シリーズより順次各機種に対応予定．

「Windows SALAD（サラダ）」（39,800円）

ワープロ，カード型データベース，表計算の3つのソフトを1パッケージにおさめたビジネスツール集．MS-Windows対応ソフトウェアであるため，アプリケーション相互に自由にデータ交換が可能．「SALAD」シリーズにはこのほかに，作図ソフト「tomato（トマト）」，通信ソフト「parsley（パセリ）」（いずれも15,800円）がある．

「Line-up（ラインナップ）」（価格未定）

RS-232Cインターフェースに専用コネクタを接続し，電話線による配線で構築できる簡易LANシステム．RS-232Cインターフェースをもつノートパソコン，デスクトップパソコンなどが最大で10台まで，プリンタが4台まで接続可能．仮想ドライブを利用して，リモートのパソコンにアクセスする機能や，アクセスレベルの設定(R/W, R/O)，パスワードの設定によるディレクトリのプロテクト，リモートパソコンの監視，制御，再起動なども可能．また，MS-Networksのエミュレーション機能をもつ．6月中に発売予定．



クリエイティブの仕事場

ナーが多い．実際の作業も，ペイントで絵を書いたり，画面のデザインをしたりということが多いようだ．

クリエイティブ部門のメンバーに共通していることは，Guideの仕事を見て，おもしろそうだと思ったことだという．実際のアプリケーションの開発において，技術的な問題が出てくれば，開発部門と打ち合わせる．利用者に近い部分のインターフェースはソフトウェア技術者ではなく，デザイナーが担当するという形態は，これからのアプリケーション開発ではいっそう増えていくのかもしれない．

最先端の情報収集にアメリカへも進出

エム・ピー・テクノロジーの組織のユニークな点としては，アメリカに現地法人をもっていることがあげられる．MPTechnologies, Inc. のスタッフは全員がアメリカ人で，日本語のできる人はほとんどいない．

社長の吉本氏によれば，アメリカのソフトウェア技術者の層の厚さは日本とは比較にならないという．Windows関連，またマルチメディア関連の情報は，アメリカのほうが日本よりはるかに豊富で早い．こうした先端情報をいち早く手に入れるというのは，アメリカ法人の大きな役割だが，今後は，アメリカ市場向け製品をどんどん開発していきたいということだ．4月にアトランタで開催されたWindows Showにも出展したという．

「アメリカのスタッフは，耳になり，目になり，手になってほしい」と吉本社長．かつて，ヒューレット・パッカード社でNewWaveの開発に携わっていた人間が責任者というだけに，アメリカ法人の今後の動向にも目が離せないのではないだろうか．Windowsという共通のプラットフォームによって，ソフトウェア開発も国際化が進みそうだ．

ワンラインNEWS
リコーは32ビットカラー液晶搭載のラップトップ型パソコン「Mr. マイツールLXII c40」を発売した．8色表示が可能で，CRTなみの鮮明さを特徴としている．

information・page

●イベント

■5月24日(金)～5月27日(月)
ハイテク東京'91
「FAシステム展」「CAD/CAM/CAEシステムショー」など、素材から設計・加工システムまでの先端・複合技術展。
会場：東京国際見本市会場／入場料：1,000円
問い合わせ先：日刊工業新聞社事業局事業部　TEL 03-3222-7239

■5月28日(火)～5月30日(木)
第73回ビジネスショウ(OSAKA)
事務処理、情報処理機器の総合展示会。
※28・29日は招待日、30日は一般公開日
会場：インテックス大阪(大阪国際見本市会場)／入場料：無料
問い合わせ先：(社)日本経営協会関西本部　TEL 06-443-6961

■6月5日(水)～6月7日(金)
ビジネスマシンショウ'91
OA機器、情報処理機器の展示会。
会場：名古屋市中小企業振興会館／入場料：無料
問い合わせ先：(社)中部経営情報化協会　TEL 052-581-6917

■6月10日(月)～6月12日(水)
ソフトウェア・シンポジウム'91
ソフトウェア技術の実践的側面に関するシンポジウム。
会場：名古屋国際会議場／参加費：40,000円(会員)、50,000円(一般)、20,000円(学校関係者)、4,000円(学生)
問い合わせ先：ソフトウェア技術者協会　TEL 03-3356-1077

■6月12日(水)～6月15日(土)
Computer Graphics Osaka '91
コンピュータグラフィックスに関するシンポジウムと関連機器の展示。
画像技術の展示会「イメージテック'91」も同時開催される。
会場：マイドームおおさか他／入場料：1,500円
問い合わせ先：(社)日本能率協会関西事業本部　TEL 06-261-7151

●海外イベント

■5月30日(木)～6月2日(日)
WEST COAST COMPUTER FAIRE
西海岸ディーラー向けの展示会。
会場：Moscone Center(米国・サンフランシスコ)
問い合わせ先：ザ・インターフェイス・グループ・インク日本支社　TEL 03-3237-8759

■6月4日(火)～6月10日(月)
台北コンピューテクス'91
コンピュータ関連の本体、周辺機器、関連製品の展示会。
会場：台北世界貿易センター展示ホール・CETRA展示ホール
問い合わせ先：遠東貿易サービスセンター　TEL 03-3407-97

●セミナー

■5月31日(金)
活用されるためのCASE―構造化手法実践編
KangaToolを使ったCASE手法を解説。
会場：YMCAホテル(東京)／受講料：30,900円
問い合わせ先：(株)ライフボート　LB ONE DAYセミナー総合案内　TEL 03-3293-4716

Microsoft University

■5月27日(月)～5月31日(金)、6月10日(月)～6月14日(金)
6月17日(月)～6月21日(金)、7月1日(月)～7月5日(金)
Microsoft Windows Programming Environment
Windowsアプリケーション開発のための入門コース。
会場：特別会場T、三田会場／受講料：150,000円(5日間)

■5月27日(月)～5月31日(金)、7月8日(月)～7月12日(金)
Microsoft LAN Manager Applications Programming
LANマネージャアプリケーション開発のための上級コース。
会場：特別会場T／受講料：180,000円(5日間)

■6月17日(月)～6月21日(金)
MS OS/2 Presentation Manager Programming Environment
OS/2 PMアプリケーション開発のための基本コース。
会場：三田会場／受講料：150,000円(5日間)

■6月18日(火)～6月19日(水)
C言語入門
Quick Cを使ったC言語の入門コース。
会場：特別会場T／受講料：60,000円(2日間)

■6月5日(水)～6月7日(金)
C言語応用
MS-C 6.0を使ってC言語の高度な技術を学ぶ実習中心のコース。
会場：特別会場T／受講料：90,000円(3日間)

■6月3日(月)～6月7日(金)
MS OS/2 Applications Programming
OS/2 APIを使った実習中心のコース。
会場：三田会場／受講料：150,000円(5日間)

■6月24日(月)～6月28日(金)
MS OS/2 Presentation Manager Applications Programming
OS/2 PMアプリケーション開発のための上級コース。
会場：三田会場／受講料：150,000円(5日間)

■6月25日(火)～6月28日(金)
Microsoft LAN Manager Programming Environment
LANマネージャアプリケーション開発のための入門コース。
会場：特別会場T／受講料：150,000円(4日間)

■7月8日(月)～7月12日(金)
Microsoft Windows Applications Programming
Windowsアプリケーション開発のための上級コース。
会場：三田会場／受講料：180,000円(5日間)

Microsoft Universityは、OS/2、Windowsアプリケーション開発者のために定期的に開催されているセミナーです。
問い合わせ先：富士ソフトウェア(株)MU教育センター　TEL 03-3452-77

NEWS & NEW PRODUCTS

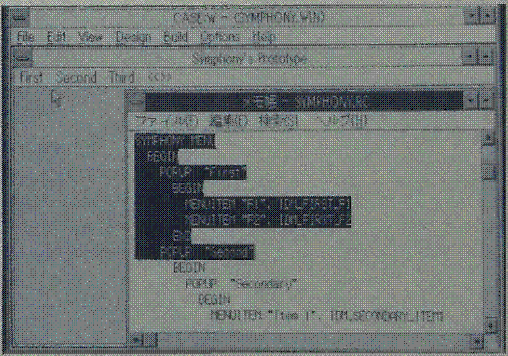
シンフォニー，Windows 3.0対応の統合開発環境「CASE:W」を発売

(株)シンフォニー(TEL 03-3443-0291)は，Windows 3.0に対応した米国CASE WORKS社のアプリケーション統合開発環境「CASE:W ver.3.0」(英語版)を発売する．従来，Windowsアプリケーションは，MS-C，Software Development Kit (SDK)に含まれるツールでMS-DOSとWindowsの環境を交互に切り替えながら開発していた．「CASE:W」では，すべてのツールを統合し，アプリケーションがWindows 3.0を制御する骨格部分のソースコードとリソースを生成するジェネレータをサポートしているため，定型的で煩雑なコードのコーディングをせずに，オリジナルなロジックの作成に専念することができる．

Windows 3.0のインターフェースをすべてサポートし，WYSWYG(What You See is What You Get)でメニュー，ダイアログボックスなどを定義していくプロトタイパー，コンパイルせずにプロトタイプの動作を確認するテストモードなどの機能を備え，プロトタイピングと同時にダイアログボックス，アイコンなどを編集することができる．また，プロトタイピングのバックグラウンドで動作し，すべての関連ファイルを生成するジェネレート機能をもつ．価格は，98,000円で，5月中旬出荷予定．

■動作環境

Windows 3.0(スタンダードモード，エンハンストモードでの使用を推奨)／拡張メモリ：5MB／ハードディスク：40MB以上(以上のハードウェア環境を推奨)



ライフボート，高速科学技術計算パッケージMathAdvantageを発売

(株)ライフボート(TEL 03-3293-4711)は，科学技術計算用のサブルーチン・ライブラリ「MathAdvantage」の販売を開始した．これは，米国QTC社で開発されたサブルーチン集で，国内総販売元であるセンチュリリサーチセンター(株)と提携・契約して発売する．「MathAdvantage」は，CADや画像処理，信号処理，データ処理，各種シミュレーションで必要となるサブルーチン約260種で構成されている．価格は，238,000円(3か月のキャンペーン期間中は198,000円)．

■動作環境

対応機種：PC-9801・FM R・PS/55・J-3100・AX／対応OS：日本語MS-DOS 3.1以上／対応コンパイラ：Lattice C 4.1以上・Microsoft C 5.1以上・Pro FORTRAN 1.2以上

メディカルリサーチイクイップメント，高速行列演算コンパイラを発売

(株)メディカルリサーチイクイップメント(TEL 03-3814-2161)は，米国アップテックシステム社の開発した高速行列演算コンパイラ「GAUSS Ver.2.1」を発売した．「GAUSS」は，フルスクリーンエディタおよびデバッガを内蔵し，dBASE，ロータスなどの外部インターフェースをもつ．2次元・3次元グラフィック機能，行列演算関数・統計関数などの豊富なライブラリ関数を備えている．また，「GAUSS」をより容易に利用するための各種アプリケーションもある．適応機種は，IBM PC，XT，AT，PS/2とその互換機で，価格は，120,000円(GAUSS)，170,000円(GAUSS 386)，220,000円(GAUSS 386VM)．

ワンラインNEWS
アップルコンピュータジャパンはパソコン用レーザプリンタと機能拡張用のアップグレード製品を値下げした．「レーザライタIINTX-J」の新価格は798,000円．

NEWS & NEW PRODUCTS

関西電機，486マシンを含むAT互換機4種を発売

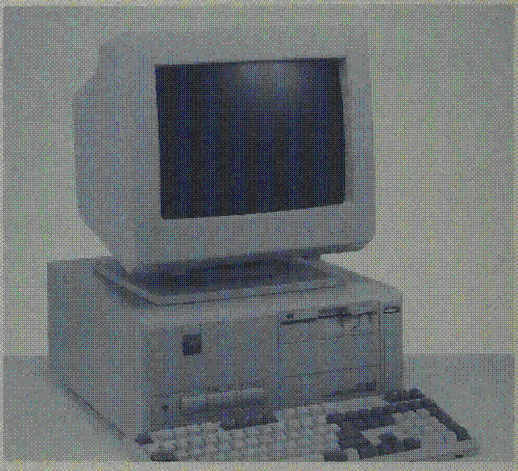
関西電機(株)プラスワン(TEL 03-5687-6912)は，CPUのパワーとコストパフォーマンスに重点を置いたAT互換機仕様の「プラスワン」シリーズを発売した．CPUに80486(25MHz)を搭載した最上位機種「486A」(写真)，「386H」(80386／33MHz搭載)，「386F」(80386SX／16MHz搭載)，「286R」(80286／16MHz搭載)の4機種がある．「486A」「386H」は標準で4MB，「386F」は2MB，「286R」は1MBのメモリを搭載している．

■486A-STD/H120　プロセッサ：80486(25MHz)／メモリ：4MB(最大64MBまで拡張可能)／外部記憶装置：5インチFDD×1(STD)・120MB HDD(H120)／拡張スロット：4／価格：598,000円(STD)，798,000円(H120)

■386H-STD/H120　プロセッサ：80386(33MHz)／メモリ：4MB(最大32MBまで拡張可能)／外部記憶装置：5インチFDD×1(STD)・120MB HDD(H120)／拡張スロット：4／価格：498,000円(STD)，698,000円(H120)

■386F-STD/H40　プロセッサ：80386SX(16MHz)／メモリ：2MB(最大8MBまで拡張可能)／外部記憶装置：5インチFDD×1(STD)・40MB HDD(H40)／拡張スロット：5／価格：248,000円(STD)，328,000円(H40)

■286R-STD/H40　プロセッサ：80286(16MHz)／メモリ：1MB(最大5MBまで拡張可能)／外部記憶装置：3.5インチFDD×1(STD)・40MB HDD×1(H40)／拡張スロット：2／価格：148,000円(STD)，228,000円(H40)



三菱電機，クライアントサーバコンピュータ「apricotシリーズ」を発売

三菱電機(株)(TEL 03-3218-3285)は，ネットワーク接続による新しい分散処理システムの構築を可能にするクライアントサーバコンピュータ「apricotシリーズ」を発売した．同シリーズは，MCA採用のPS/2互換機であり，DOS/Vを採用する．80486を搭載した「FTサーバ」，80486と80386の「Qi(キィ)ワークステーション」「LANターミナル」の4モデルがある．セキュリティ機能を標準装備し，標準ネットワークOSとしてNetWare，LANマネージャを採用する予定．

■FTサーバ(FT486-25S)　プロセッサ：80486(25MHz)／メモリ：4MB(最大16MBまで拡張可能)／外部記憶装置：3.5インチFDD(1.44MB)×1，347MB HDD(M3507-A103)・647MB HDD(M3507-A106)・1047MB HDD(M3507-A110)／拡張スロット：7／価格：3,980,000円～5,980,000円／HDDは最大5.2GBまで拡張可能，1.2GBのストリーミングテープ(オプション)が内蔵可能

■Qiワークステーション(Qi486-25)　プロセッサ：80486(25MHz)／メモリ：4MB(最大16MBまで拡張可能)／外部記憶装置：3.5インチFDD(1.44MB)×1，100MB HDD(M3505-A110)・200MB HDD(M3505-A120)・305MB HDD(M3505-A132)／拡張スロット：4(32ビット×2・16ビット×2)／価格：1,980,000円～2,700,000円

■Qiワークステーション(Qi386SX-16)　プロセッサ：80386SX(16MHz)／メモリ：1MB(最大5MBまで拡張可能)／外部記憶装置：3.5インチFDD(1.44MB)×1，50MB HDD(M3502-A105)・100MB HDD(M3502-A110)／拡張スロット：16ビット×3／価格：770,000円・930,000円

■LANターミナル(LS386SX-16)　プロセッサ：

日本電気は9600bpsの伝送速度に対応した全二重モデム「DATAX・SP9624」を発売した．CCITT V.42，V.42bis，マイクロコム MNPクラス5に対応している．

NEWS & NEW PRODUCTS

80386SX(16MHz)／メモリ：1MB(最大8MBまで拡張可能)／外部記憶装置：3.5インチFDD(1.44MB)×1・50MB HDD／価格：625,000円／イーサネットI/Fを標準装備

日本IBM，POWERステーションの最高速モデルなど2機種を発売

日本アイ・ビー・エム(株)(TEL 03-3586-1111)は，同社の高性能ワークステーション「IBM RISCシステム／6000」(POWERステーション)シリーズに，サーバ専用の最上位モデル「POWERサーバー950」，と卓上型の新モデル「同320H」を発売した。「950」は，現行機にくらべて処理能力を60%向上した業界最高速のサーバ専用機で，現行の最高速機「550」の41MHzのRISCプロセッサを移植し，56.3SPECmark，25.2MFLOPSの高い処理能力をもつ。「320H」は，25MHzのRISCプロセッサを搭載し，32.4SPECmark，1.7MFLOPSの処理能力をもつ高性能卓上型モデル。

コンテック，NetWare対応LANアダプタシリーズ34種を発売

(株)コンテック(TEL 03-3769-1061)は，機種，伝送媒体，接続方法別に開発したNetWare対応のLANアダプタボード・ユニットなど，全34種の「LAN HELPER」(総称)を発売した。「LAN HELPER」は，ノベル社のネットワークOSの日本語版NetWare 386 v3.1Jに対応し，異機種間接続を可能にするマルチベンダ仕様で，国内のほとんどの機種に対応している。98NOTE，DynaBook用の「NOTE LANアダプタ」(写真)は，パソコンのI/O拡張ユニットを使用せずに，拡張バスコネクタから直接LANに接続することができる。LANアダプタはノンインテリジェント型で，カスタムLSIの採用で低消費電力を実現，従来はボード上のスイッチで行っていた設定を不要とした。



日本IBM，ISDN対応などのネットワーク関連製品を発表

日本アイ・ビー・エム(株)(TEL 03-3586-1111)は，端末からネットワーク全体の資源を効率的に利用し，クライアントサーバコンピューティングを効果的に実現するための通信関係製品として，以下の製品群を発表した。

PS/55，複数ワークステーションの制御装置IBM3174で，ネットワーク上のプログラムやデータに自在にアクセスでき，ネットワークの構成変更を容易に行える「APPN」(拡張対等通信ネットワーキング)製品として，OS/2用のソフトウェア「SAAネットワーク・サービス／2」，IBM3174制御装置のマイクロコード「構成サポートC」「APPN機構」。

音声・データ・イメージなどの情報をディジタル信号によって同一ネットワークで伝送するISDN対応製品として，OS/2に対応したISDNアダプタ「IBM ISDNインターフェース・アダプター／A」(PS/55用)と「IBM 3174 ISDNアダプター」(3174制御装置用)。

ネットワーク管理者が，ネットワークを通じて汎用コンピュータまたは単一のサーバからネットワーク全体へアプリケーション，データなどを送り，導入・更新することができる「NetView分散管理プログラム／2」(DOS・OS/2対応)，「同リリース3」(MVS用)。11月までには全製品を出荷の予定。

ワンラインNEWS
米クレイ・リサーチ社は年内に16個のCPUを搭載した世界最高水準のスーパーコンピュータを発売することを表明した。

NEWS & NEW PRODUCTS

マイクロソフトウェア，SUN上でDOS用ソフトを実行するツールを発売

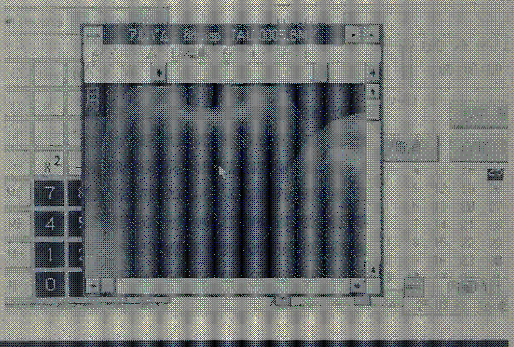
マイクロソフトウェア(株)(TEL 03-3818-7888)は，SUN SPARCstationのSUN OS上でIBM PC/ATの環境をソフト的に実現し，MS-DOSで動作するソフトウェアをSPARC上で動かすことができるMS-DOSソフトウェアエミュレータ「SoftPC」を発売した．「SoftPC」は，SUN SPARCstation-2をはじめ，SPARC互換機での使用が可能．MS-DOS3.3がインストール済みで，SUN OS/V4.0.3に対応している．価格は，280,000円．

レンガ，Windows 3.0対応アプリケーション2種を発売

(有)レンガ(TEL 03-3494-1098)は，Windows 3.0に対応したアプリケーション2種を発売した．

「WINDOWS BACKGAMMON」(WIN3対応版)は，2人対戦モードをサポートしており，価格は9,800円．サイトライセンス版98,000円もある．

「WINDOWS TOOLS」(WIN3対応版)は，Windowsの操作性の向上を目的としたツール集で，スクリーンの写真化を行う「カメラ機能」，画像データを収録する「アルバム機能」(写真)，ファイルに見出しをつけて整理する「コメント機能」，クリップボードのデータをダイレクトに印刷する「プリント機能」など20種の機能がある．価格は，68,000円．OS/2版78,000円もある．



マイクロソフト社など21社が先進的プラットフォームACE構想を提唱

Microsoft，Compaqなどの有力企業21社が共同で，新しい先進的なプラットフォーム「Advanced Computing Environment」(ACE)構想に取り組むことを発表した．この構想には，The Santa Cruz Operation社の統合UNIXシステム「Open Desktop」，Microsoft社の「OS/2 3.0」，x86ベース，MIPS社の縮小命令セット(RISC)型マイクロプロセッサベースのコンピュータに関する新しい標準仕様が含まれている．また，ネットワーク環境との接続性も提供する．この構想に基づく製品は，1992年にユーザーに対して提供される．

AX協議会，VGA機能をもつビデオシステムの仕様を決定

AX協議会(TEL 03-3440-9100)は，AX仕様にビデオシステムの標準規格「VGA」をサポートした「AX-VGAシステム」の仕様を策定し，その発表を行った．

従来，AX独自のビデオ規格として，日本語モードでの漢字などの特殊性を考慮した「EGA」互換のビデオシステム「JEGA」(640×480ドット)をサポートしてきたが，米国の標準が「VGA」へ移行したことなどから，AXの新しいビデオシステムとして，日本語・英語両モードともに640×480ドットの表示能力をもった「AX-VGAシステム」を決定した．同システムには，現行と同様にハードウェアで日本語環境をサポートした「AX-VGA/H」と，日本語環境をソフトウェアのみで実現した「AX-VGA/S」の2種のシステムがある．

また，「AX-VGAシステム」の仕様は，同協議会の思想である“オープン・アーキテクチャー”に基づいて，業界・ユーザーに公開していく．

Programmer's Program

MS-Cパワーユーザーのために ソースコードやツールを提供する

Microsoft C Developer's Toolkit(DTK)

OS/2プログラミングの参考資料としても価値がある

「Microsoft C Developer's Toolkit」は，MS-C 6.0のパワーユーザー向けに発売されたツールキットで，略してDTKと呼ばれる．「Microsoft C Run-Time Library Source Routines」「Professional Advisor」「Programmer's Toolkit」の3つで構成されている．

ランタイムライブラリのソースコード集は，MS-C 5.1でも公開されていたが，MS-C 6.0版の「Microsoft C Run-Time Library Source Routines」では約400のソースコードが提供されている．このソースルーチンには，浮動小数点演算とグラフィックス，OS/2マルチスレッド，BIOS，一部の低レベルルーチンを除く，MS OS/2，MS-DOS用の全ランタイムライブラリのソースコードが含まれている．

ライブラリ関数を機能拡張することができるし，完成されたプログラミング例として参考にすることもできる．とくに，OS/2のプログラミングについては，現時点では参考となるプログラミング例が少ないだけに学習用として最適ではないだろうか．

「Professional Advisor」は，ヘルプデータベース中のヘルプファイルを，アプリケーションから利用するためのヘルプエンジンである．ヘルプデータベースは，QuickHelpフォーマットで作られたテキストファイルを，HELPMAKEでエンコードすることにより，ユーザー自身が作成することができる．QuickHelpフォーマットは，トピックのテキスト，フォーマット，リンク，そして情報を制御するコードをもっている．

ヘルプエンジンに用意されているヘルプデータベースから索引を取り出したり，リンクの張られている項目の中身を取り出すファンクションなどを，アプリケーション側から呼び出すことによって利用できる．

「Programmer's Toolkit」には，CodeView用のシンボリックデバッグ用オブジェクトモジュールフォーマットやオブジェクトモジュールフォーマット（OMF）の拡張部分の情報，デバッグ情報用に拡張された.EXEフォーマットや.ILKと.SYMファイルのバイナリフォーマットなど，MS-C 6.0の各種ファイルのフォーマット，ソースブラウザ情報，Programmer's WorkBenchの拡張機能が収められている．

ライブラリソースコードだけで2.5MBのディスク容量が必要

DTKをビルドするには，ハードディスクが必要である．ライブラリソースが2.5MBのディスクスペースを占有するうえ，ソースのコンパイルやライブラリの作成のために，余分なディスクスペースを確保しなければならない．また，Advisor LibraryとProgrammer's Toolkitで約0.5MBのディスクスペースが必要になる．

「Run-Time Library Source Routines」のインストールは，SETUPプログラムにしたがって，コピー先のディレクトリ名を指定し，ディスクを入れ換えるだけでできる．このSETUPプログラムは，ライブラリソースと，その関連ファイルを，ハードディスク上に設定されたディレクトリ（およびSETUPプログラムが作成するサブディレクトリ）にセットアップする．同じ名前のディレクトリがすでに存在する場合は，内容をすべて消去しておく必要がある．指定されたディレクトリが存在しない場合は，自動的に作成される．

「Professional Advisor Library」や「Programmer's Toolkit」には，特別なセットアッププログラムはなく，ハードディスクにそれぞれ適当なディレクトリ（たとえば，ADV，BSC）を作成し，個々のディスクの全ファイルをコピーするだけである．

ソースコードのコンパイルはBLDLIBSコマンドで行う

ライブラリソースは，4つのメモリモデル（スモール，ミディアム，コンパクト，ラージ）および，ふたつのモ

ード（MS-DOS，MS OS/2）で，それぞれコンパイルすることができる．オブジェクトには，MS-DOS，MS OS/2のどちらでも動作するものと，オペレーティングシステムごとに，別々のファイルとなっているものがある．どのメモリモデル用のオブジェクトを作成する場合も，メモリモデルおよびオペレーティングシステムに依存するオブジェクトが，同時に生成される．倍長整数ヘルパルーチンも，オブジェクトを作成する過程でコンパイルされる．

SETUPでコピー先に指定したディレクトリから，BLDLIBSコマンドを起動して，ソースをコンパイルし，オブジェクトファイル（*.OBJ）とライブラリファイル（*.LIB）を作成する．BLDLIBSコマンドの書式は，

BLDLIBS ［〈memory model〉］

である．引数〈memory model〉には，コンパイルするライブラリのメモリモデルを指定する．メモリモデルの引数には，s（スモールメモリモデル），m（ミディアムメモリモデル），c（コンパクトメモリモデル），l（ラージメモリモデル），all（全メモリモデル）を指定することができる．なお，倍長整数演算のヘルパ（LIBH.LIB）は，全メモリモデルで作成できる．

BLDLIBSバッチファイルは，MS-DOSおよびMS OS/2モード用の機能ごとのライブラリを作成する．新しく結合ライブラリを作成する場合は，MS-C 6.0に含まれるSETUPプログラムを起動し，「/LIB」オプションを使って行う．なお，結合ライブラリを作成するには，MS-C 6.0に含まれる浮動小数点ライブラリが必要である．

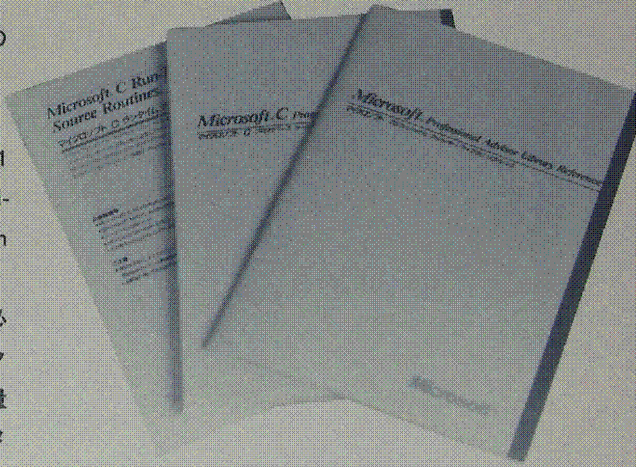
パッケージは，ディスク4枚，マニュアル2冊などの簡素な構成となっている．しかも，技術サポートは受けられないということなので，十分にMS-C 6.0について理解している「真のプロフェッショナル」のためのツールであるといえる．

●Microsoft C Developer's Toolkit

販売：マイクロソフト（株）／価格：50,000円（MS-C 6.0のパッケージの中にある申込券での注文のみ可能）

【必要機器】

本体：MS-C 6.0に対応した機種／OS：MS-DOS Ver.3.1以上もしくは，OS/2 Ver.1.1A以上／対応コンパイラ：Microsoft C Professional Development System Version 6.0，Microsoft Macro Assembler 5.1以上／必要メモリ：640KB以上（MS-DOS），空きメモリ1MB以上（OS/2）／必要ディスク装置：1MB容量の1台以上のフロッピーディスク装置とハードディスク装置（MS-Cインストール後の空き容量3MB以上）／推奨機器：1MB以上のLIM Ver.4.0対応EMSメモリ（MS-DOS），1MB以上のキャッシュディスク（MS-DOS），マウス



オンラインソフト採集帳 6
バイナリファイルの差分出力更新ユーティリティ BDIFF/BUPDATE

オンラインソフトの多くは，パッケージソフトにくらべ頻繁にバージョンアップが行われている．これは，バグフィックスだけでなく，リアルタイムに利用者の声を生かした機能拡張などが行われるためである．

新しいバージョンのソフトをダウンロードして復元してみると，ドキュメントと拡張子にBDFをもったファイルが出てくることがある．このファイルが，BDIFFによって出力された差分ファイルである．

BDIFF123.LZHをLHAで復元してみると，ドキュメントを含むいくつかのファイルが出てくる．差分ファイルを出力するBDIFF，差分ファイルから旧ファイルをアップデートするBUPDATE，ほかに，プログラム中のRCSヘッダをスキャンし，リビジョンをタイムスタンプに設定するMODTIMEや，旧バージョンのBDIFFなどで作成したBDFファイルを処理可能な形式に変更するためのBDFMODである．

バイナリファイルの差分を出力する「BDIFF」

BDIFFは，2つのファイルを比較して差分を出力するユーティリティだ．BDFファイルの名前が指定されていない場合は，新しいバージョンのファイルの拡張子をBDFに変更したものがカレントディレクトリに出力される．

たとえばTEST.EXEと，このファイルの旧バージョンであるTEST.ORGがカレントディレクトリに存在した場合，これらのファイルから差分を生成するには，DOSのコマンドラインから

```
A>BDIFF TEST.ORG TEST.EXE
```

と入力し，[RETURN]キーを押すと，BDFファイルTEST.BDFが出力される．

オプションは，BDFファイルに複数の差分を追加する-A，逆方向へのファイルの更新情報をもたせることによって「バージョンダウン」を可能にする-Xなどが指定できる．くわしくは，BDIFF.DOCに解説されている．

差分からファイルを更新する「BUPDATE」

BUPDATEは，BDFファイル（BDIFFによって作成された差分ファイル）を使用して，ファイルのアップデートを行うためのユーティリティである．また，複数の差分ファイルがまとめられているBDFファイルについて，その内容を表示したり，一部の差分を削除することも可能である．

BDFファイル内には，アップデート処理の対象となるファイル名が記述されているので，単にアップデートを行うだけの場合は，BDFファイルを引数に取るだけで処理を行うことが可能である．たとえば，旧バージョンのTEST.EXEと差分ファイルであるTEST.BDFをカレントディレクトリにおいて，DOSのコマンドラインから

```
A>BUPDATE TEST
```

と入力し，[RETURN]キーを押すと，対象ファイルTEST.EXEに対して差分の更新が行われ，新バージョンのTEST.EXEが新規に作成される．複数の差分がBDFファイルに記述されている場合も，一括して更新することが可能だ．

また，更新対象のファイルについては複数のファイル名を記述したり，ワイルドカードで指定することもできる．「*.EXE」と指定された場合には，BDFファイル中の拡張子がEXEのファイルのみがアップデートされる．

なお，アップデートに使用するファイルは，BDIFF実行時の旧ファイルとタイムスタンプ・チェックサム・長さが同一である必要がある．タイムスタンプの比較を避けたい場合には，-Iオプションを付加する．また，BDIFFの実行時に-Iオプションを付加して作成したBDFファイルの場合は，このオプションのあるなしにかかわらずタイムスタンプの比較は行われない．

-Vオプションを指定すると，BDFファイルの内容（更新対象となるファイル名，作成された日時，ファイルサイズ，リビジョンおよび差分ファイルのサイズ）が表示される．-Dオプションは，複数の差分が格納されているBDFファイルから，特定の差分を削除したいときに指定する．ファイルで示されるファイルパターンにマッチする差分が削除される．-Xオプションは，アップデートの実行を指定する．各オプションが省略された場合は，このオプションがデフォルトとして指定される．-Rオプションは逆アップデートを行う場合に指定する．ただし，このオプションを使用するためには，BDFファイルがBDIFFの-Xオプション付きで作成されている必要がある．デフォルトでは，アップデートすると対象ファイルの拡張子はORGという名前に

ワンラインNEWS
日立製作所はパソコンやWSのパッケージソフトを開発・販売する子会社「日立アプリケーションシステムズ」を設立し，関連会社のソフト販売を一元化する．

変更されるが，-Oオプションを指定するとすべての更新の対象となったファイルは消去される．-Wオプションは，ワークディレクトリを指定する．

*

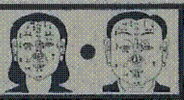
バイナリファイルの差分更新ユーティリティであるBUPDATEを使うことにより，従来のファイルから，差分ファイルだけで，手軽に新しいバージョンに変更することができる．最近はオンラインソフトも多機能化して，数100Kバイト以上のものもめずらしくはなくなってきたが，これを使えば通信時間もディスクスペースも大幅な節約になる．

■BDIFF　分類：ファイル差分出力・更新ユーティリティ／ソフト名：BDIFF/BUPDATE／登録名：BDIFF123.LZH／著作権者：PSH（T. Tanaka TNT technology）／動作環境：MS-DOS／入手場所：日経MIX／復元方法：LHARC E BDIFF123／配布条件：単体であること，内容の変更がないこと，メディア代を除いて無償であること，配布によって著作者が一切の制限を受けるものではないことを条件として再配布は自由となっている．

ファイル構成

```
BDIFF123.LZH
Name          Original   Date      Time
------------  --------  --------  --------
BDFMOD.EXE       18928  90-02-09  01:02:00
BDIFF.DOC        16249  90-02-09  01:23:00
BDIFF.EXE        24320  90-02-09  01:23:00
BUPDATE.EXE      24326  90-02-09  01:23:00
MODTIME.EXE      13216  89-08-03  01:02:00
------------  --------  --------  --------
   5 files       97039  90-03-22  09:08:28
```

★UPDATE INFORMATION★　「LHA」（4月号で紹介）のバージョンが2.12に（3月21日・NIFTY-Serve）

HEALTHY?  **UNHEALTHY?**

⑥「オタクデブ」にさようなら

「パソコンオタクはデブである」というイメージが定着してしまった．オタクではなく，日々の暮らしのためにプログラムを書いている人たちも，デブの宿命から逃れられないということが少なくない．原因は，運動不足とストレス食い．

「(身長−100)×0.9」という式は，適正体重を算出するためのものとしてよく知られているが，絶対的に正しいわけではない．なぜなら，肥満とは体重のうちに脂肪が占める率で決められるからだ．スポーツマンで筋肉質な人とおなかのでた人とでは，同じ身長でも適正体重が違う．とはいえ，スポーツをして脂肪を燃焼させて，ひきしまった身体づくりをしよう……なんて非現実的なことをすすめるつもりはない．第一，スポーツで燃焼するエネルギーなんてたかがしれているのだ．死ぬ思いでフルマラソンを走っても2千数百キロカロリー，体脂肪にして300グラムあまりにしかならない．

では，どうすればいいかというと，食生活を工夫するのがいちばん効果的．糖質，蛋白質，脂質とビタミン，繊維などをバランスよく……なんていっても簡単ではないから，日常的に食品のカロリーを判断する習慣をつけるだけでだいぶ違う．まず，てんぷら，フライなど，衣をつけて油で揚げた食べものは避けたほうがいい．植物油は身体にいいなんていわれているが，それはコレステロールなどの面だけでいえること．カロリーは純粋な植物油のほうが高い．サラダが身体にいいというのも迷信に近い．生野菜は食べた感覚のわりにあまり量をとれない．植物油を使ったドレッシングは高カロリーだ．また，宅配で便利なピザにも，蛋白質と脂肪のかたまりであるチーズがたっぷり使われている．

もちろん，そうした食品をぜんぜん食べるなというわけではない．そういう知識をもっていれば，外食をするときにもある程度選択をすることができる．それで，焼き魚定食などのカロリーを抑えた和食を意識的に選ぶ機会が増えれば，それだけで効果があることもある．和食は，カロリーコントロールに向いている．食べて満腹感のあるご飯に，副食として低カロリーのものを組み合わせるようにする．

SE-NET拾い読み

4月号からディスクサービスがはじまって，SE-NETへのアクセスは大幅に減る，と予想していたのだが，そんなことはなく，相変わらずにぎわっています．ひと月あきましたが，「SE-NET拾い読み」の登場です．掲載にあたり，一部修正・割愛させていただきました．

「プログラマの内輪話」ボード

■眠りバグのお話

いやはやしんどかったぁ／ Sunvisor
平成03年03月19日（火） 午後05時13分35秒

今日、僕が作った給与計算プログラムがバグを出しました。作ってから3年も使っているのに、これまで、一度も出たことのない症状がでて大騒ぎでした。(騒いでいたのは給与担当と私の二人だけでしたが)
市販のアプリと違って、狭い職場である程度決ったパターンでしか使わないプログラムにはこの手の「眠りバグ」はつきものですが、開発からだいぶ時間も経っているので、原因がわかってから開発時の頭に戻すまでに苦労しました。あ〜まじめにドキュメント書いておくんだった。(いつもこんなことばっかり言ってる)
Sunvisor

眠りバグといえば……／Otaku
平成03年03月20日（水）午後03時49分38秒

発売からすっかり歳月が経過し，従ってサポート期限も切れ（おそらく），その上開発元まで倒産（プログラマの解散？）してしまったソフトウェアのバグはどうなるのでありましょう。ちなみにこれは，シリコンビーチ・ソフトウェアという米国（そう。あの強大にして，世界のＭＰにして，国際的体育会……）のソフトウェアメーカのMacintosh用ソフトウェアで［シリコンエクスプレス］のことでした。［スーパーペイント］はきちんと日本でサポートされてるようだけど、さ。
Otaku

■素人の乱入

素人の乱入／Taco
平成03年03月29日（金）午前05時32分35秒

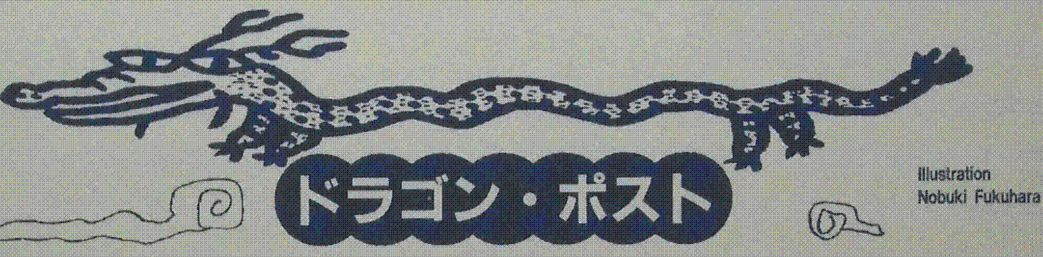
あの、ここは、本職のプログラマでないと書込めないのでしょうか。
私は通りすがりのアマグラマのTacoです。
私の勤務先は、コンピュータとは関係の無い業種です。社員数30名ほどの小さい会社なのですが，IBMのAS400が置いてあります。端末は2台つながっております。伝票発行業務にしか使っていません。
パスワードなどはデフォルトのままなので、私は勝手に自分のユーザープロフィールを登録して、仕事が暇なときに中身を覗いています。誰も咎めないのです。
パソコンと違って、どんな操作をしてもハングしないのがすごいじゃん、でも反応がちょっとトロいな、………などとやっているうちに、やっぱり何かプログラムを書きたいな、なんて思うわけです。
ところが、使える言語が、RPG400という、聞いたことのないヤツで，よく解らん訳です。時々メンテにくるお兄さんに尋ねたら、“ああ、COBOLみたいなもんですよ。”と、教えてくれました。もちろん、私は、名前を聞いたことしかありません。
“この、CL（制御言語）ってなんですか？”
“ああ、それはBASICみたいなかんじかな。”
で、そのCL（制御言語）ちうヤツでやったろかいな、と考えてマニュアルをひろげると、ぜーんぜんパソコンのBASICと違うんだな、ここれが。
何が何やらちんぷんかんぷん。
カタログを見ると、PASCALやＣもあるらしいのだけれど、ただ遊ぶためだけに買ってはもらえないし。
こういうシステム関係の入門書は一般の書店にも置いていないし。
そもそも、AS400ってメジャーな機械なのですか？　Taco

前の会社で使ってました／Rei
平成03年03月30日（土）午前01時34分50秒

AS/400（正確にはこう書く？／）って……IBM系列ではゆーめいだと……（^_^;)・・。玲★

こうしたんだ／Taco
平成03年04月07日（日）午前04時52分24秒

私の会社でのAS/400(こう書くのが正しかったです。ごめんなさい。)の導入の経緯についてですが，なるべくしてなったのです。
実は、IBMは我が社のお得意様なのです。で、IBMの人が、たまたま訪ねてこられたりしたときに、他社のコンちゃんがでんと座っていたのでは具合が悪い訳ですね。少なくとも、我が社のエライ人はそう思ったみたいです。
導入したのは、ちょうどTVや新聞で森繁さんと太宰さん(この人がタコ社長)の広告がバンバン出ていた頃です。設置時にきていた代理店(IBMでは、こういう風には呼ばないらしい）のお兄さんたちも、
「こんな機械だったのかぁ」
「小さいよなあ」
なあんていいながら、英文のマニュアルをひっくり返して四苦八苦してました。
Taco



ドラゴン・ポスト

Illustration
Nobuki Fukuhara

この欄は，読者からのお便りを中心に構成します．本誌に対するご意見・ご感想などをお待ちしております．鬼より恐い読者からの声ということで，「ドラゴン・ポスト」です．本誌についているアンケートはがきにご意見の欄を設けてありますから，そこに書いてください．ペンネームでも歓迎です．ただし，ハガキには本名と住所をしっかり書いてくださいね．掲載誌をお届けします．

★何か似ているな？

ところでエプソンのPC CLUBって何かに似ていませんか？ボクはPC-8001に似ていると思うのですが……ねだんも同じ168,000円．でも3.5inchFDつきだし，速いし……だいぶちがうかな～．時代の流れって早いですね．ボクは何をかくそうPC-8001のユーザーです．（景山）

――あら～！PC-8001ね．それは，気づかなかった．なんたって1979年発売のマシンですよ．「ユーザーです」ということはまだ現役なんだろうか？

★アップデートがこんなに早いなんて！

Turbo PASCAL 6.0が発売になり，私のところへもアップデートサービスの案内が届きました．5.5プロ版を買ってまだろくに使っていないうちに，はや6.0！アップデート代20,394円は高い！少し待ってバグの取れた頃に安く買った方がいいと見送りを決めました．このあたりの状況を特集して欲しいものです．（手塚　朗）

――そうなんですよ．ボーランドのこのごろのアップデートのテンポにはびっくりさせられます．去年，米国でTC++がでて，年末には日本版．と思うまもなく，米国でボーランドC++の発売です．Windows用のPASCALもでたんですって，いずれ日本版が発売されるでしょうね．

★「遊」工房は楽しい

フロッピーが付録になったこともあり，創刊から表紙の面白さもあって見ていましたが，なかなかむずかしい内容らしくてとっきにくかった．プログラム「遊」工房のように遊び心のあるところがいいと思います．（中井）

――作者の谷口さんは，毎回アイデアだしに大変なようです．なるべく楽しいプログラムを作りたいけど，ちょっと凝るとソースが長くなってしまうし……というところが悩ましいとか．掲載プログラムを発展させて，おもしろいものを作ってまたお便りください．

★とじ込み付録を取ることできずに……

４月号を購入したのは，記事を読みたかったのはもちろんですが，１番の理由はとじ込み付録の〝Turboシリーズ″が目当てでした．ところが切り離そうとしたところ，フロッピーディスクの封筒にくっついて取ることができず使いものになりませんでした．せっかく付録をつけてくださるのですから，もう少し考えて提供してほしいと思いました．（吉野）

――申し訳ありませんでした．ほかにもこのようなお便りがたくさん……．４月号では，はじめてフロッピーディスクをつけたので，なるべく確実に読者のお手元にまで届くようにと，ノリ付けにしたのですが，トラブルが多かったようです．５月号からは改善されていると思いますので，これに懲りずにご愛読をお願いいたします．

★LHaにつられて……

LHaにつられて買ったというのが正直な理由です．ビンボーな大学生は通信なんて問題外です．でんわさえないのに！（DAi）

――LHAは，ファイル圧縮プログラムの標準といえるほど普及しているLHarcの上位バージョンです．LHarcは，米国でも広く使われている日本産の国際ソフトウェアです．通信をしていなくても，ファイルバックアップのときなどにディスクスペースを有効に使うことができます．でも，本文もちゃんと読んでくださいね．

★私はプログラム初心者だ！

はっきりいって，私はプログラムは初心者だ．しかもプログラムするのは苦手だ，しかし，この本は読みものとしてはおもしろいので既刊分はすべて買った．でも，内容には理解不能な部分もあるので，こんな読者用のコーナーもつくってほしい．それと本の値段のわりに読者プレゼントが質・量ともにヒサンではないだろうか．（渡部）

――えー，１～３号の創刊記念プレゼントでは，ハンディ98を出したりしてかなり気張ったんですが……．４号からは付録フロッピーディスクというコストのかかることをやってしまったんで……悪しからず．

★先は長いですね

ハードディスクをつけたばかり．コンパイルスピードの格段のアップにニコニコです．しかし本体640KBは狭い．先は長いですね．（鵜木）

――いーなー！容量はどのくらいですか．このごろは，10万円もだせば80MBくらいのが買えるし，最近は180MB，1MBキャッシュ付きで20万円を切るなんてものもあります．このあとはメモリですか．お金かかりますね～．

付録ディスクのご案内

このフロッピーディスクに含まれる実行ファイル，ソースリストおよびテキストファイルは「月刊Programmer's Page」1991年6月号に掲載されたものです．

付録のフロッピーディスクは，5"2HD（8セクタ／トラックMS-DOS）でフォーマットされています．3.5"や2DDなどへのコンバートサービスは行っておりません．なお，オンラインサービスSE-NET（03-5276-1297）でも，同じ内容のファイルを用意しています．

フロッピーディスクの内容は次の通りです．本誌およびディスクに含まれるドキュメントファイルをよく読んでからご利用ください．

●付録ディスクについての説明

ディレクトリ名：ルート（¥）

収録ファイル名

README.DOC（テキストファイル）

●特集・研究「プログラミング統合環境」

③MS-C 6.0の統合開発環境（p.18）

ディレクトリ名：TOKU3

収録ファイル名

PTES2.C　PTES2.PXT

PTES2.MAK

④MS-DOSの開発環境／QuickCの使い方（p.33）

ディレクトリ名：TOKU4

収録ファイル名

GRAPHDSP.C　GSAMPLE.MAK

GRAPHDSP.H　GDSPHLP.TXT

GSAMPLE.C　GSAMPLE.EXE

⑤Turbo C++／Borland C++のIDE（p.45）

ディレクトリ名：TOKU5

収録ファイル名

PPF.BAT　PPF_SUB.C

●プログラム『遊』工房（p.76）

タイピング練習ゲーム「TYPEGAME」

ディレクトリ名：PLAY

収録ファイル名

TYPEGAME.C TYPEGAME.EXE

●実践C塾（p.82）

ディレクトリ名：JUKU

収録ファイル名

LIST1.C　LIST3.C

LIST2.C

●MS-DOSツール・ラボ（p.92）

簡易16進電卓「EXPR」

ディレクトリ名：TOOL

収録ファイル名

LIST1.ASM　EXPR.ASM

LIST2.ASM　EXPR.EXE

LIST3.ASM　KAIJO.ASM

●98活用アセンブラ技法（p.101）

ディレクトリ名：98ASM

収録ファイル名

BACK2.ASM

●綴じ込み付録

ディレクトリ名：DATA

収録ファイル名

MENU.TXT（テキストファイル）

●オンラインソフト採集帳

本誌2月号で紹介したファイル管理ユーティリティ「FD」（9801版），および4月号で紹介した高圧縮書庫管理ソフト「LHA」がバージョンアップされましたので収録しました．

ディレクトリ名：ONLINE

収録ファイル名

LHA212.EXE

FD98_230.LZH

FD98_23C.LZH

注意：オンラインソフトは，そのソフトウェアの性格上，該当月号が発売されるまでに新しいバージョンが発表されている可能性があることをご了承ください．また，サポートは基本的にパソコン通信のみで行われておりますので，ご注意ください．

●クイズ・ランダムスクリーン（p.142）

クイズ解読ツール「PUZLE」

本誌掲載のクイズ・ランダムスクリーンの画像をディスプレイ上に再現し，マウスを使って並び替えることにより，解答を得ることができるユーティリティです．対応機種は日本電気のPC-9801シリーズです．詳しい使い方はPUZLE.DOCを参照してください．

ディレクトリ名：OMAKE

収録ファイル名

6月号の問題

CONFIG.SYS　RNDSCREN.GI

AUTOEXEC.BAT　RNDSCREN.BI

PUZLE.EXE　RNDSCREN.EI

PUZLE.DOC　RNDSCREN.RGB

RNDSCREN.RI

5月号の解答

HANSEN.RI　HANSEN.EI

HANSEN.GI　HANSEN.RGB

HANSEN.BI

Windows 3.0用 WallPaper（壁紙）

HANSEN.BMP

＊使用上の注意

本誌掲載プログラムおよびデータを利用するには，MS-DOSの基礎知識が必要です．また，このディスクにはMS-DOSのシステムは入っていません．別途MS-DOSが必要です．

＊お知らせ

今月号のオンラインソフト採集帳で紹介した「BDIFF」は，作者の希望により付録ディスクには収録されておりません．大手商用NETまたはSE-NETで入手していただくようお願いします。

バーチャルスペースガイド

P子　某ソフトハウスで企画営業を担当している．26歳．
C平　某メーカーのシステムソフトウェア開発部門に所属．25歳．

P子　私，このごろ半村良に凝っちゃってんのよ．伝奇小説ってやつ．

C平　ふ～ん，伝奇小説ね．それでこのごろは，「花とゆめ」じゃなくて文庫本もって歩いているわけだ．

P子　「産霊山秘録（むすびやまひろく）」とか，いろいろあるけど，「石の血脈」がおもしろかったわ．古代から現代まで，歴史のあやしげな部分や怪談みたいのものが全部登場するんだから．

C平　巨石文明とか，ドラキュラ伝説とか，暗殺教団とか，サンジェルマン伯爵とか……．

P子　なんだ，知ってんじゃない．つまらない．でも，あれで歴史に興味もっちゃったわよ．このあいだの週末に，巨石を見たくて奈良までいってきちゃった．益田岩船ってすごいのね．突然あらわれるし，大きいし．

C平　益田岩船については，松本清張も「火の回路」という小説を書いてたよ．あれは，ゾロアスター教の祭壇だという仮定だったかな．

P子　わー，それ読んでみたい．絶対にあれは不思議だよ．知りたいことが増えてしまうよ～．

C平　実は，暗殺教団の山の長がいたというアラムートの砦を取材したビデオをもっているんだ．ずーっと前にNHKでやっていた．

P子　げっげっ！お願いだよ．見せておくれよ！

C平　ところで，半村良のペンネームの由来は知っている？

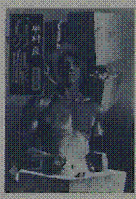
P子　……？

C平　イーデス・ハンソン．

■石の血脈

半村良
角川文庫

建築家の失踪事件にかかわった石川は，市民社会の裏にうごめく特殊な組織の存在を知る．建築家の行方を追い組織と戦ううちに，石川自身が組織に取り込まれてしまう．そこで，語られる闇の人類史では，人類発生の秘密，不老不死の「病」など，数かずのなぞが解きあかされる．古代から現代にいたる歴史の謎を織り込みながら，それをひとつの物語にまとめあげており，いっきに読んでしまうおもしろさがある．



620円

読者プレゼント

1 Turbo Pascal 6.0(PC-9801)



1名

提供:(株)ボーランドジャパン

2 バスマウス (PC-9801)



3名

提供:(株)エム・ビー・テクノロジー

応募は本誌に添付のアンケートはがきをご利用ください．（1991年6月17日消印有効）．なお，応募者多数の場合は，抽選になります．当選者発表は発送をもって代えさせていただきます．

3 マイクロソフト特製めんこ



10名

提供:マイクロソフト(株)

4 マクロアセンブラプログラマーズテキスト

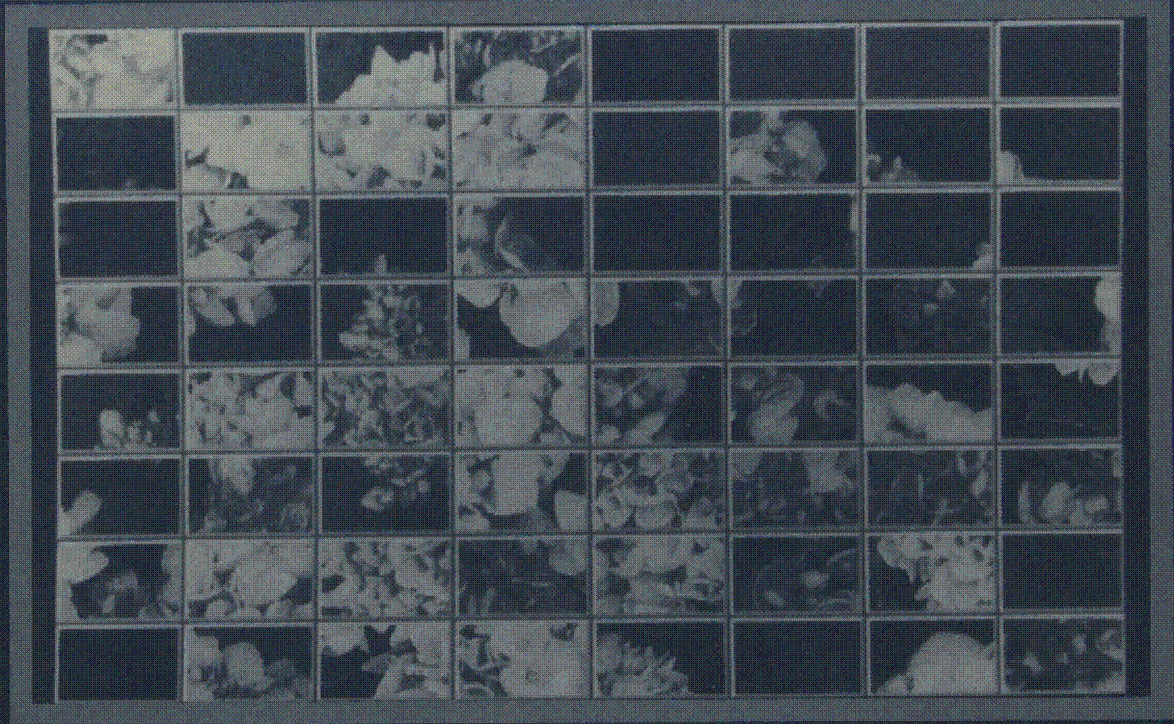


3名

クイズ・ランダムスクリーン

ある1枚の画像を64分割し，ランダムに並べ替えてあります．
さて，もとの画像は何を写したものでしょうか．

ヒント：梅雨時に一番似合う花



正解者の中から30名様に本誌表紙イラスト（日暮修一画）入り月刊プログラマーズ・ページ特製オリジナルマグカップをさしあげます．

応募方法：本誌に添付のアンケートはがきをご利用ください．

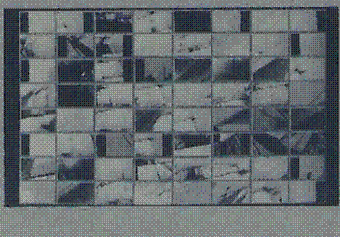
締め切り：1991年6月17日（消印有効）

なお，当選者の発表は発送をもって代えさせていただきます．

＊画像データは，付録ディスクやBBSでもサービスしています．

5月号の問題と正解

【問題】



【正解：帆船または日本丸】



月刊プログラマーズ・ページ オンラインサービス（SE-NET）のお知らせ

付録にフロッピーディスクをつけたため，ディスクによるプログラムサービスは中止いたします．オンラインサービスのBBSは，多数の読者の登録をいただいておりますので，今後も継続いたします．編集部とのコミュニケーションの場としてご利用ください．

●BBSへの接続

SE-NET　03-5276-1297

①接続するとIDの入力を求めてきますので，GUESTと入力しリターンキーを押してください．
②お知らせを表示後，メインメニューのコマンドプロンプトが表示されます．
③月刊プログラマーズ・ページへのご意見などは「ドラゴンポスト」ボードに書き込んでください．
④ファイルの転送が終了したら，0と入力して回線を切ります．

Programmer's Page MONTHLY MAGAZINE FOR PROGRAMMERS

月刊プログラマーズ・ページ バックナンバーのご案内

■定価・各980円







創刊号
1991年1月号
特集・プログラマの仕事場

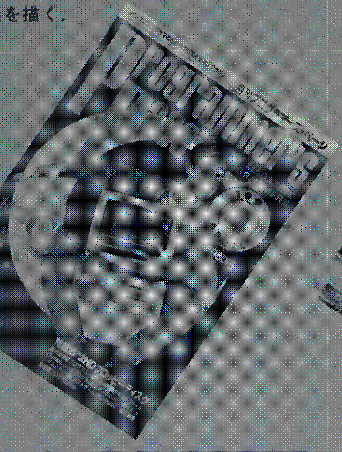
パーソナルコンピュータでソフトウェアを開発する現場の姿をさまざまな角度からレポートし，これまであいまいなイメージしかもたなかったパソコンプログラマ像を描く．

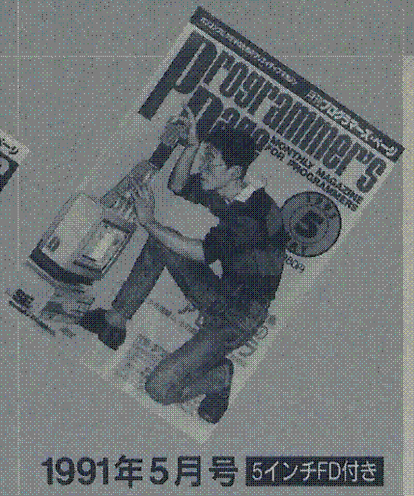
1991年 2月号
特集・Windowsの時代である！

Windows 3.0の発売で話題の渦中にある新しいOS環境の開発システムを徹底解説．Windowsのサンプルプログラムを関数ごとに図をつけて紹介する．

1991年3月号
特集・この言語を使いなさい!!

数あるプログラミング言語の中から14をピックアップ，それぞれの言語を使い込んだプログラマが，その特性から使い方のポイントまでを解説する．





1991年4月号 5インチFD付き
特集・C読本

パソコンプログラミング言語の標準的存在になったC言語ととことん付き合うための大特集．入門からオブジェクト指向への流れまでを紹介する全５章．

1991年5月号 5インチFD付き
特集・基礎力としてのアセンブラ

プリミティブなプログラミング言語でありながら，それ故に無視することができないアセンブラ．コンピュータを理解し，その能力をひきだすためにもう一度アセンブラをみなおす．

「月刊プログラマーズ・ページ」のバックナンバーは，ご注文いただければ，全国どこの書店でもお求めになれます（書店に届くまで１か月ほどかかります）．
小社に直接ご注文なさるときは，何月号かを明記のうえ，現金書留にて雑誌価格＋送料を下記までお送りください．（１部の場合の送料は260円です．２冊以上の場合は，号数・冊数により送料が異なりますので，小社にお問い合わせください．）

〒102 東京都千代田区平河町
2-4-14 平河町KSビル4階
(株) 翔泳社
マーケティング部PP係宛
TEL 03-3263-0447

Programmer's Page
MONTHLY MAGAZINE FOR PROGRAMMERS

月刊プログラマーズ・ページ7月号
（6月18日発売）

付録：5″ 2HDフロッピーディスク付き!!

特集
最新デバイスドライバ研究

パソコンの拡張性を支えるソフトウェアインターフェース，それがデバイスドライバである．この特集では，MS-DOS，OS/2，UNIXそしてMS-Windowsにいたる最新のOS環境におけるデバイスドライバの特徴と作成法について，具体的な実例をあげながら徹底的に解説する．

月刊プログラマーズ・ページ　1991年6月号
1991年6月1日発行

発行所　株式会社翔泳社
東京都千代田区平河町2-4-14平河町KSビル4F・5F
TEL 03-3263-0447　〒102
発行人　長廻健太郎
編集長　清水　隆
副編集長　鈴木嘉平
編集スタッフ　杉野貴美子＋上田純美礼＋芳山充弘
デザイン　SE（翔泳社デザイン研究所）
広告スタッフ　相澤正夫＋澤本良知＋種村嘉彦
協力　根本裕三郎＋(株)ドモン・マインズ＋佐藤裕美
＋小林弘人(図版デザイン)
写植　株式会社東巧社＋株式会社トーヨー・プリント社＋株式会社帆風
印刷　凸版印刷株式会社
定価　980円（本体951円，税29円）

EDITOR'S PAGE

●6月号である．1月号で創刊して以来，本誌も無事(?)半年をすごすことができた．「もう半年」「まだ半年」，人により違うだろう．パソコンの世界も同じだ．「もうあれがでてしまった」「まだあれがでてこない」，いろいろである．連休を前にした春の日々，風に舞う桜の花もせわしく感じられる．「山桜　枝切る風もさりげなく　花をさながら　我がものとする（西行）」という世界は遠いかな．(TA)

●最近のボーランドのバージョンアップ攻勢には，鬼気迫るものがある．ついこの間Turbo CがTurbo C++になったかと思っていたら，すぐにBorland C++が現れ，Turbo Pascalが6.0になったなと思う間もなくTurbo Pascal for Windowsが出た．TPWはTP6.0とは別系統の製品とはいうものの，こうたて続けに出されたのではたまらない．はっきりいって，バージョンアップ貧乏である．もうじきParadoxも出るし……．ところで，今月号の「HEALTHY? UNHEALTHY?」に出てくる“オタクでぶ”って誰のことだ？（ピザが好き／嘉平）

●某S社のビデオ編集機を買った．ところが同じS社の製品なのに，出力側のビデオ端子と接続するケーブルがない．結局，赤外線コントロールしかできないとのことで，5,000円也の出費．せめて同じ会社の製品くらいサポートして欲しい……．しかし，いつの間にか，ステレオミニミニだのコントロールSだの，聞いたこともない端子がたくさん出ていた．こうして「機械オンチ」になっていくのでしょうか．(ま)

●今回の特集で，日本アイ・ビー・エム基礎研究所の鈴木所長にお話をうかがうことができた．せっかくの機会なので鈴木氏の著書「研究者のテイスト」（共立出版）を読んでみた．1988年に1年間かけて雑誌に連載された内容をまとめたものだそうだが，いま読んでもその内容は十分に新しい．年々技術の進歩がめざましいコンピュータの世界だが，やはり専門の研究をしている方は，1歩も2歩も先を歩んでいるのだということを痛感した．本誌で研究者の方にお話をうかがったのは，今回がはじめてだが，読者の皆様はどんな感想をお持ちだろうか？（kos)

●連休を前に「あ～あ，私もオバンの仲間入りか」とつぶやく．普段，全く年齢のことなど忘れている（考えないようにしている？）僕ではあるが，年に2回どうしても思い出させてくれる．その1日が4月30日なのである．「若いね」と言われる年齢を過ぎ去った僕にだって，多少無理をすれば，先日のことのように鮮明に思い出せるのに（嘘かも知れない），「何故，二十歳でオバンなの？」と聞きたい．とにかく「誕生日，おめでとう／」（既に過ぎてるって?!）（よ）

愛読者カードご記入のお願い

本誌をお買い上げいただき、誠にありがとうございます。
お手数ですが、今後の誌面づくりの参考にさせていただきたいと存じますので、
各欄にご記入の上、ご返送くださるようお願い申しあげます。

月刊プログラマーズ・ページ READER'S SURVERY CARD 愛読者カード

本誌の号数　　年　　月号

本誌を知った理由
□1. 書店店頭　□2.他社出版物（　　　）
□3.小社出版物　□4. DM　□5.口コミ

本誌を購入した書店　　市（区）　　書店

パソコンの所有形態（機種名）
□1. 個人所有　（　　　）
□2. 職場にある（　　　）
□3. 持っていない

よく使用するソフトウェア

よく使用するOS

今月号でおもしろかった記事（いくつでも）

今後取り上げてもらいたい特集のテーマ

貴方が選ぶベスト3

パソコン雑誌	①	②	③
ソフトウェア	①	②	③
パソコン本体	①	②	③

本誌についてのご感想、お気づきの点、その他どのようなことでも結構ですので、ご意見をお聞かせください（「ドラゴン・ポスト＝読者の声」欄に掲載させていただく場合もあります。ペンネームをご希望の方はあわせてご記入ください）。

月刊プログラマーズ・ページ 広告資料請求カード

8	会社（学校）名		9	所属（学部）		10	従業員数	

※下記の該当する番号をご記入ください。

11	業種		12	職種		13	パソコン利用状況	

業種　①製造業　②金融・保険・不動産業　③商社・卸・小売業
④建設業　⑤通信・運輸　⑥システム・ソフト
⑦その他サービス業　⑧公務員　⑨その他

職種　①コンピュータ（ハード）技術者　②コンピュータ（ソフト）技術者　③エレクトロニクス技術者
④プログラマ　⑤その他の技術者　⑥専門職
⑦事務・営業職　⑧経営管理職　⑨商工サービス
⑩教職　⑪学生　⑫その他

パソコン利用状況　①会社で利用　②個人で利用　③会社でも個人でも利用　④利用していない

●希望される広告資料　　年　　月号

広告資料請求No. / 希望項目												
A 購入希望	A	A	A	A	A	A	A	A	A	A	A	A
B 説明希望	B	B	B	B	B	B	B	B	B	B	B	B
C 資料希望	C	C	C	C	C	C	C	C	C	C	C	C

定期購読のご案内

綴じ込みの振替用紙を使って、本誌の定期購読をお申し込みになれます。
1年間12冊=11,760円(本体11,417円/税343円)
郵送料サービス

郵便はがき

料金受取人払

麹町局承認

3908

差出有効期間
平成4年11月
30日まで

切手不要

102-00

東京都千代田区平河町2-4-14
平河町KSビル

株式会社 翔泳社
マーケティング部 行

月刊プログラマーズ・ページ
READER'S SURVEY CARD　愛読者カード

氏名	性別 □1男 □2女	生年月日 19　年　月　日
住所　〒		電話 (　　) −

職業			ユーザー区分
□1ソフト技術者	□2ハード技術者	□3技術系会社員	□1 プロ
□4事務系会社員	□5営業系会社員	□6教師	□2 マニア
□7一般公務員	□8経営管理職	□9理系学生	□3 一般ユーザー
□10文系学生	□11その他(　　)		

プレゼント記入欄	クイズ・ランダムスクリーン回答欄
希望商品番号	答

郵便はがき

料金受取人払

麹町局承認

3910

差出有効期間
平成4年11月
30日まで

切手不要

102-00

東京都千代田区平河町2-4-14
平河町KSビル

株式会社 翔泳社
月刊プログラマーズ・ページ
広告資料請求係行

＊当社で記入

1	SE	2	受付No.

月刊プログラマーズ・ページ広告資料請求カード

3	会員番号	

＊すでに会員番号をお持ちの方はご記入ください。

4	フリガナ 氏名	姓　名	5	生年月日	19　年　月　日

			資料送付希望先の番号に○印をつけてください。
6	自宅住所	〒　電話　(　　)	
7	会社住所	〒　電話　(　　)	

払込票

通常払込料金加入者負担

口座番号 東京 4-154493

加入者名 株式会社 翔泳社

金額 11760 円

※ 払込人住所氏名

備考

受付局日附印

切り取らないで郵便局にお出しください。

記載事項を訂正した場合は、その箇所に訂正印を押してください。

払込通知票

通常払込料金加入者負担

口座番号 東京 4-154493

加入者名 株式会社 翔泳社

金額 11760 円

払込み 料金 特殊

備考

受付局日附印

※(郵便番号)

TEL( )

払込人住所氏名

この払込通知票は、機械で使用しますので、下部の欄を汚さないよう特に御注意ください。また、本票を折り曲げたりしないでください。(郵政省)

各票の※印欄は、払込人において記載してください。

ト株式会社

ームをはじめ、ミニコン、
などのシステムプログラム
レティングまで、幅広く手
こ。1970年に設立以来、高
大規模プロジェクトに参画
AIシステムやUNIX、Σ
理など新しい技術やシステ
マイクログラフィックス
gner」をはじめ、優れた
グローバルな業務展開に
のソフトハウスと一線を画
る女性が担当。編集部は、
社に総務課課長武車美登里

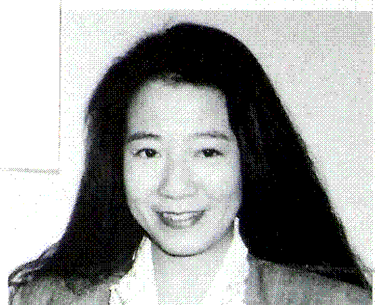
ちろん、そういう目的意識ある方
大歓迎です。ただパソコンとか機
手に仕事をしていますと、往々に
自分本位になり、自分だけの意見
し通すという方も少なくありませ
グループ作業が多いものですから、
うまくやっていける素質のある方
いと。」

――温和な性格というと、抽象的ですけど、そういう性格の判断は採用の段階で可能ですか？

「私どもでは電話でお申し込みいただいた時点から言葉遣いや、自分の思ったことを正確に口で言えているかどうかということをチェックしていますし、性格テストも導入しています。また中途採用する場合には、会社を何回変わっているか、前の会社では何が不満で辞めているか、ということは一通りお聞きしています。時に本人のわがまま

がメーカーの新機種や新システムを開発することがメインですので、生半可な知識がある方より、逆に全く知識がなく一から勉強する方のほうが、熱心で素直に覚えてくれる場合も多いんです。また将来プロジェクトリーダーとして上に立って才能を発揮してもらうことを考えた時には、性格的なものが重要になってきますからね。」

――ただ多少人間的にギスギスしていても、目標に向かって努力するガッツのあるタイプもいると思うんですが。



〈プロフィール〉
武車美登里（むしゃみどり）

東京都池袋出身。大妻女子短期大学。化学工業メーカーで営業職として4年勤務後、そのメーカー出資によるコンピュータショップに移籍。そこで、ワープロインストラクターの資格を取得、3年の経験を積む。そこで培った幅広い知識と人間関係を生かし、同社に入社。現在、総務部総務課課長として人事・経理などを幅広く担当。入社6年目、33歳。

定期購読のご案内

■定期購読は、1年間12冊と限定させていただきます。

■購読料のお支払は、この振替用紙をご利用ください。銀行振込、現金書留によるご送金は、ご遠慮ください。

■振替用紙に、ご住所、お名前をご記入のうえ、郵便局の貯金窓口にてお支払ください。

■小社より領収書の発行はいたしませんので、郵便局にて「郵便振替払込金受領証」をお受取ください。

■お申し込みいただいた方には、弊社より直接発送いたしますので、到着が発売日より遅れることがございますが、ご了承ください。

月刊プログラマーズ・ページ　定期購読直接申し込み書

この欄は、加入者あての通信にお使いください。

Programmer's Page MONTHLY MAGAZINE FOR PROGRAMMERS

年　　月号より

年間購読料 11,760円(税・送料込み)

■ご希望の購読開始月号をご記入ください。

■送付先（住所・氏名は正確にご記入ください）

住所	□□□-□□	フリガナ	
		氏名	
		TEL	

この払込通知票は、機械で使用しますので、下部の欄を汚さないよう特に御注意ください。また、本票を折り曲げたりしないでください。（郵政省）

郵便はがき

102-00

東京都千代田区平河町2-4-14
平河町KSビル

株式会社 翔泳社
月刊プログラマーズ・ページ
広告資料請求係行

2	受付No.	

マーズ・ページ広告資料請求カード

*すでに会員番号をお持ちの方はご記入ください。

名	5	生年月日	19　年　月　日

□10文系学生　□11その他（　　　　）　ユーザー

プレゼント記入欄	クイズ・ランダムスクリーン回答欄
希望商品番号	答

6	自宅住所	〒　電話（　　）	資料送付希望先の番号に○印をつけてください。
7	会社住所	〒　電話（　　）	

綴じ込みの1

人材募集
広告企画

ちょっと気になるこの会社

ジスクソフト株式会社

ジスクソフトは、メインフレームをはじめ、ミニコン、ワークステーション、パソコンなどのシステムプログラムを、企画・設計・開発・コンサルティングまで、幅広く手がけてきた先進のソフトハウスだ。1970年に設立以来、高度な技術力で常に時代の先行く大規模プロジェクトに参画してきた実績は、現在では対話型AIシステムやUNIX、Σ計画、音声言語処理、日本語処理など新しい技術やシステム開発に生かされている。また、マイクログラフィックス社のWindowsソフト「Designer」をはじめ、優れた海外ソフトの日本への紹介など、グローバルな業務展開にも意欲的に取り組んでいる。他のソフトハウスと一線を画す同社は、採用・教育に関しても女性が担当。編集部は、青山に本社オフィスを構える同社に総務課課長武車美登里さんを訪ねてみた。

温和で協調性に富む人材を希望

——現在、貴社ではどのように人材を採用しているのでしょうか？

「今はどこのソフト会社にも言えると思いますが、仕事はたくさんあるのに人がいなくて困っている状況ですね。ですから私どもの会社では新規も中途も特に募集に関しては限定していません。ただ採用する場合に最も重視しているのは性格ですね。温和で性格のいい方を希望しています。」

——温和な性格というと、抽象的ですけど、そういう性格の判断は採用の段階で可能ですか？

「私どもでは電話でお申し込みいただいた時点から言葉遣いや、自分の思ったことを正確に口で言えているかどうかということをチェックしていますし、性格テストも導入しています。また中途採用する場合には、会社を何回変わっているか、前の会社では何が不満で辞めているか、ということは一通りお聞きしています。時に本人のわがままということもありますから、正当な理由かどうかを見極めるようにしています。」

——中途採用の場合、何回も転職している人は、やはり厳しいチェックの対象になるのですか？

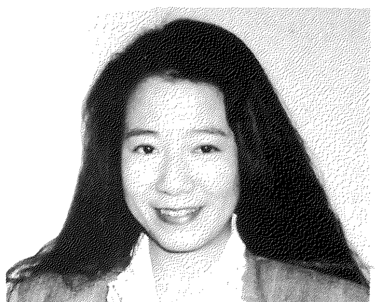
「そうですね。３回以上転職されている方や、あまりにも短期間で会社を変わっている方は採用から外れますね。」

——全くの未経験者でも採用されることもあるのですか？

「はい。というのも当社の仕事の内容がメーカーの新機種や新システムを開発することがメインですので、生半可な知識がある方より、逆に全く知識がなく一から勉強する方のほうが、熱心で素直に覚えてくれる場合も多いんです。また将来プロジェクトリーダーとして上に立って才能を発揮してもらうことを考えた時には、性格的なものが重要になってきますからね。」

——ただ多少人間的にギスギスしていても、目標に向かって努力するガッツのあるタイプもいると思うんですが。

「もちろん、そういう目的意識ある方なら大歓迎です。ただパソコンとか機械相手に仕事をしていますと、往々にして自分本位になり、自分だけの意見を押し通すという方も少なくありません。グループ作業が多いものですから、人とうまくやっていける素質のある方でないと。」



〈プロフィール〉
武車美登里（むしゃみどり）

東京都池袋出身。大妻女子短期大学。化学工業メーカーで営業職として４年勤務後、そのメーカー出資によるコンピュータショップに移籍。そこで、ワープロインストラクターの資格を取得、３年の経験を積む。そこで培った幅広い知識と人間関係を生かし、同社に入社。現在、総務部総務課課長として人事・経理などを幅広く担当。入社６年目、33歳。

――それは協調性、情緒安定性みたいなものですね。

「そうです。それに優れた技術力が加われば言うことはないのですけれど。」

社員旅行の企画も研修のうち

――今現在、社員数はどれぐらいですか？

「今年、５名新入社員が入りまして、60名程になりました。その５名のうち４名は文系出身です。」

――教育についてはどのような内容になっていますか？

「入社前の３月中旬に導入研修を行っています。これは強制するものではなく、研修所を借りた２泊３日の寝泊まりを通じて新卒者同士のコミュニケーションを図ってもらうことが目的です。入社後、４月いっぱいは本社内で、コンピュータの概念やワープロの総合研修などがあります。そして５月以降は各職場の内容に合ったソフトの構築を、３～５年ぐらいの経験ある先輩社員からOJTというかたちで行います。また５、６月には自己紹介をワープロと写植機で社内報というかたちで作ってもらったり、社員旅行の企画をしてもらいます。自分たちでミーティングしたり、旅行会社と交渉する中で、同期意識やお客様との折衝能力を磨いてもらうわけです。」

――貴社ではアメリカの会社との取引きもありますが、英語力についてはどうでしょう？

「社内的に語学研修はありませんが、昨年社員１人がアメリカへの長期出張に当たり、日本の社団法人の海外訓練教会の３週間の研修を受けています。今後は英会話に意欲ある人には研修を取り入れていきたいですね。ただ語学が苦手だとか意欲のない社員には負担になりますから、強制するようなことはしたくはありません。」

――では中途採用の場合の教育は？

「技術の下地がある方には即戦力として配属を決めて、うちではこういうかたちで仕事をやってますという風に教えます。一方、ソフトの未経験者には配属してからいろんなマニュアルを与えながら、簡単なソフトを作ってもらい、徐々に覚えていってもらいます。専門学校卒業者の中には前の会社でオペレーター的な仕事しかやらせてもらえなかったから、ぜひソフトを作ってみたいという人もいるんです。そういう人にもやりがいを見出せるように仕事を与えています。」



ジスクカラーは個人が持てばいい

――貴社ではフレックスタイム制を導入するなど、まさに柔軟な勤務体系を採っているようですが、武者さんから見て社内の雰囲気はどんな感じですか？

「のんびりしてると思います。当社では納期間近以外は残業もほとんどありませんし。それに仕事がアプリケーションソフト等の開発ではなく、３～５年の長いタームのプロジェクトが多いものですから。中途で入ってきた人がよくこんなにのんびりで大丈夫なの、と驚くぐらいなんです。」

――そういう長期の仕事で、じっくり取り組む貴社の姿勢は、やはり社長さんの方針なのですか？

「そうですね。やはり人が一番の財産だという考えに立っています。ただ受注だけに頼っていると会社としての成長も限界がありますから、最近は海外ソフト関係の代理店業務に事業を拡大していますし、将来的には受注との割合を半々ぐらいに持っていきたいと考えています。」

――社員間のコミュニケーションはどのように図られているのでしょう？

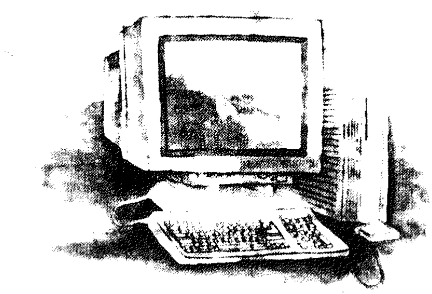
「当社には役職を含まない10数名の社員による職場委員会がありまして、ボウリング大会やソフトボール大会を自発的に行っています。また野球やサッカーの同好会もありまして、会社からも補助金を出しています。中途採用者の中にはカラオケを歌わせるとマイクを離さないとかユニーク（？）な人も大勢いるようです。

ただ当社では会社のカラーを押しつけるようなことはしていません。ジスクカラーはその人個人が持てばいいという考えです。例えばアメリカがそうですが、自分がソフトを開発した場合はその人個人に著作権が入ってくるようにうちの社長も考えていますし、いい意味での個人主義はこれからも取り入れていきたいですね。」

人事担当者に女性を配属する会社は日本ではまだそれほど多くはないだろう。ジスクソフトはその点でもかなり先進的な会社だといえる。仕事に関しても個人を大切にする欧米的な良さを取り入れているが、一方で協調性を求めるなど、日本的なチームワークの良さも残している。ソフト開発がボーダーレスに向かう時代を一歩先んじて柔軟に押し進めようとする同社の姿勢に、将来の可能性を感じる人も多いのではないだろうか。

株式会社エルテックス
ELTEX

常にパーフェクトでありたい。



ユーザーや上司が求めているものを機械的に作り上げていく。そんな顧客志向型が、本当のＳＥ・プログラマーと呼べるであろうか。常に思考し続け、より高い価値観を徹底追求し、期待以上のものを創造していく。それが私たちの本当の姿なのだ。提案型ソフトウェアハウスのエルテックスは、そんなＳＥ・プログラマーたちの集まり。たえずパーフェクトなものを求め続けていく集団なのです。

●

人材募集

会　社　概　要

設立　昭和60年12月14日

資本金　1,000万円

売上高　2.3億円（平成１年度）

事業内容　パソコンを中心としたシステムの提案・開発・販売
①CD-ROM･LDなど新しいメディアに対応したシステムの開発
②ネットワークの端末システムの開発
③特定業種向けCADの開発・販売
④基本ソフトの開発

特徴　大手クライアント向けのシステム開発が多く、ネットワーク関連システムやマルチメディアなど幅広く対応している
また、CADを中心とした自社プロダクツにも積極的で、その比率も増加している
基本ソフトの分野では、ハードウェア専門のベンチャー企業とジョイントし、OEM製品のBIOS・ユーティリティなどを企画から手掛けている

取引先　三菱電機、三菱自動車、日本航空、旭硝子ほか

関連会社　株式会社ソーテック、学校法人福井情報ビジネス総合学院

募　集　要　項

職種　①システムエンジニア　②プログラマー

資格　専門卒以上20～36歳の方
C言語・アセンブラ言語経験者大歓迎
未経験でも好奇心・向学心の旺盛な方は歓迎します

給与　17万円（専門卒初任給）～46万円（36歳課長、諸手当含む）
24歳例：23万4千円（情報処理１種手当・住宅手当含む）
30歳例：31万6千円(中級SE、情報処理１種･住宅手当含む）

手当　時間外手当
役職手当（5千円～6万円）
住宅手当（1万5千円）
資格手当（2種：5千円～特種：2万円）
通勤手当

賞与　賞与年2回（平成１年度実績年間5か月）
決算賞与年１回（平成１年度実績１か月）

勤務地　本社（横浜駅近く）
◇派遣・出向や社外での開発作業はありません

勤務時間　フレックスタイム制（コアタイム10：00～16：00）
試用期間中は、標準時間9：00～17：45

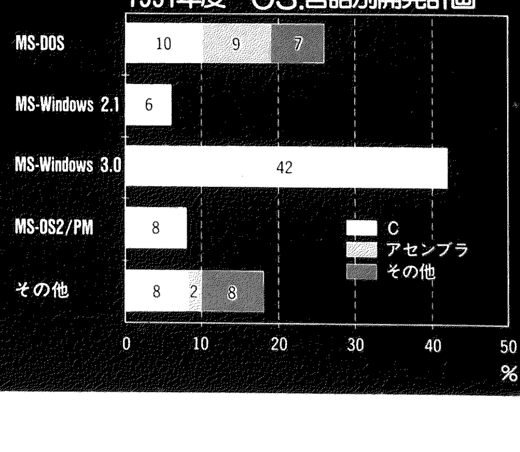
休日休暇　完全週休2日制（土日)、祝日、年3回連休（GW、夏期、年末年始)、有給休暇、特別休暇

福利厚生　社会保険完備、退職金制度、財形貯蓄、社員旅行、海外研修、定期健康診断、社員寮（制限有)、スポーツクラブ法人会員

応募方法　電話連絡の上、履歴書(写真貼付)、職務経歴書を持参し来社してください　郵送も受け付けます

連絡先　〒220　横浜市西区南幸二丁目20番5号　東伸24ビル5階
Tel(045)322-9661　総務課

1991年度　OS.言語別開発計画



統合開発環境メニュー一覧

メニュー内の下線付きの文字は，［GRPH］キーとの組み合わせによりショートカットキーとして使うことができる．
「→」はサブメニュー（＊）があることを示し，「...」付きのメニューは選択するとダイアログボックスが表示される．

■PWB Version 1.10 (MS-C 6.00A)

●File

New	新規ファイルの作成
Open...	既存ファイルのオープン
Merge...	ファイルの挿入
Next	コマンドラインで指定された次のファイルをオープン
Save	編集中のファイルの保存
Save As...	編集中のファイルを名前を変えて保存
Save All	すべてファイルを保存
Close	編集中のファイルのクローズ
Print...	ファイルの印刷
DOS Shell	DOSシェルの起動
Exit	PWBの終了

●Edit

Undo	直前の編集操作を取り消す
Redo	Undo処理直前の復元
Repeat	直前の編集コマンドを繰り返す
Cut	選択したテキストをクリップボードに移動
Copy	選択したテキストをクリップボードに複写
Paste	選択したテキストをクリップボードから挿入
Clear	選択したテキストをすべて削除
Set Anchor	カーソル位置をアンカーとして記録
Select to Anchor	カーソル位置からアンカー位置までを選択
Box Mode	ボックス，ストリーム，またはライン選択モードを設定
Read-Only	ファイルの変更を禁止
Set Record...	記録するマクロ名の設定
Record On	マクロ記録の設定・解除
Edit Macro	〈record〉疑似ファイルの作成

●View

Split Horizontal	ウィンドウを水平に分割
Split Vertical	ウィンドウを垂直に分割
Size Window	アクティブウィンドウの大きさを変更
Maximize Window	ウィンドウの大きさの切り換え
Close Window	アクティブウィンドウのクローズ
Compile Results	エラーウィンドウのオープン

●Search

Find...	テキストの検索
Selected Text	選択されているテキストの検索
Repeat Last Find	同じテキストをもう一度検索
Change...	テキストの置き換え
For File...	ディスク上のファイルを検索
Next Error	次のエラーを含む行を検索
Previous Error	前のエラーを含む行を検索
Set Error	カーソル位置のエラーを含む行を検索
Go to Mark...	定義済みのマーク位置を検索
Define Mark...	ファイル内のマークの位置を定義
Set Mark File...	マークを含むファイルのオープン

●Make

Compile File:	メニューに表示されているファイルのコンパイル
Build:	メニューに表示されているプログラムのビルド
Rebuild All	プログラムリスト中のすべてのモジュールを再ビルド
Build Target...	プログラムリスト中の指定したターゲットのビルド
Set Program List...	プログラムリストの設定，または新規作成
Edit Program List...	プログラムリストのファイルを修正
Clear Program List	現在のプロジェクトを終了

●Run

Execute:	最後にビルドされたプログラムの実行
Command Line...	プログラムのコマンドラインの実行，またはデバッグのために設定
Debug:	カレントプログラムのデバッグ
Run DOS Command...	DOSの任意のコマンドを実行
Customize Menu...	メニューに項目を追加

●Options

Environment...	PWB環境変数の設定
Key Assignments...	キーにコマンドを割り当てる
Editor Settings...	エディタスイッチの変更
Build Options...	buildプロセスの設定
C Compiler Options...	コンパイラオプションの設定
Browse Options...	PWB Source Browserデータベースの設定
LINK Options...	LINKのコマンドラインを設定
NMAKE Options...	NMAKEのコマンドラインを設定
CodeView Options...	CodeViewオプションの設定

●Browse

Goto Definition...	シンボルの定義を検索
Goto Reference...	シンボルのリファレンスを検索
View Relationship...	シンボルの関係を表示
List References...	変数のリファレンスを表示
Call Tree...	プログラム呼び出しツリーを表示
Outline...	プログラムアウトラインを表示
Next	次の定義かリファレンスに進む
Previous	前の定義かリファレンスに戻る
Case Sensitive	シンボルの検索で大文字と小文字を区別する
Split Window	ブラウザウィンドウに画面を分割する

●Help

Index	トピックのアルファベット順リスト
Contents	ヘルプの目次を表示
Topic	カーソル位置の単語のヘルプトピック
Help on Help	ヘルプシステムの使い方のヘルプ
Next	次のヘルプトピック

■Microsoft CodeView Version 3.00A

●File

Open Source...	ソースファイルのオープン
Open Module...	モジュールグループからモジュールをオープン
Print...	指定したテキストの印刷
DOS Shell	DOSシェルの起動
Exit	CodeViewの終了

●Edit

Undo	最後の編集操作の取り消し
Copy	選択されたテキストをクリップボードに複写
Paste	クリップボードのテキストを挿入

●View

Source	新しいsourceウィンドウのオープン
Memory	新しいmemoryウィンドウのオープン
Register	registerウィンドウのオープン
8087	8087ウィンドウのオープン
Local	localウィンドウのオープン
Watch	watchウィンドウのオープン
Command	commandウィンドウのオープン
Help	helpウィンドウのオープン
Output	プログラム出力の表示
Maximize	カレントウィンドウを最大にする
Size	カレントウィンドウの大きさを変更
Close	カレントウィンドウのクローズ

●Search

Find...	式の検索
Selected Text	テキストの検索
Repeat Last Find	同じテキストをもう一度検索
Label/Function...	ラベルまたは関数を検索

●Run

Restart	メモリにプログラムを再ロード
Animate	プログラムをトレースしながら実行
Set Runtime Arguments...	コマンドライン引数の設定
History On	デバッグセッションをファイルに保存
Undo	ヒストリファイルをさかのぼります
Replay	記録したセッションのリプレイ

●Watch

Add Watch...	ウォッチ式の設定
Delete Watch...	ウォッチ式の削除
Set Breakpoint...	ブレークポイントの設定
Edit Breakpoints...	ブレークポイントの編集・削除
Quick Watch...	式の評価

●Option

Source Window...	sourceウィンドウの表示を変更
Memory Window...	memoryウィンドウの表示を変更
Trace Speed...	プログラムの実行速度を設定
Language...	式の評価言語を設定
Screen Swap	画面のフリップ・スワップを設定
Replay...	ヒストリのリプレイ条件を設定
Case Sensitivity	記号の大文字・小文字の区別を設定
386 Instructions	80386フォーマットで表示

●Calls

Callsメニューは，プログラムの状態に応じて内容が変化する

●Help

Index	ヘルプの索引を表示
Contents	ヘルプの目次を表示
Topic:	特定のトピックのヘルプを表示
Help On Help	ヘルプシステムの使い方を表示

■Microsoft QuickC Version 2.00

●F/ファイル

N/新規	新規ファイルの作成
O/読込...	既存ファイルのオープン
F/再読込	直前にエディットしたファイルのオープン
M/追加...	ファイルの挿入
S/保存	編集中のファイルの保存
A/名前を変えて保存...	編集中のファイルを名前を変えて保存
V/すべて保存	すべてのファイルを保存
P/印刷...	ファイルを印刷
D/MS-DOSコマンド	DOSコマンドの実行
X/終了	QuickCの終了

●E/編集

U/元に戻す	最後の編集操作を取り消す
T/カット	選択したテキストをクリップボードに移動
C/コピー	選択したテキストをクリップボードに複写
P/ペースト	クリップボードのテキストの挿入
E/削除	指定されたテキストの削除
R/編集禁止	ファイルの変更を禁止

●V/表示

S/ソースファイル...	ソースファイルの表示
I/インクルードファイル...	インクルードファイルの表示
O/実行画面	プログラム出力の表示
Z/拡大	アクティブウィンドウを最大または元のサイズに戻す
W/ウィンドウ	表示するウィンドウの制御

●S/サーチ

F/検索	テキストの検索
S/指定文字列検索	選択されているテキストの検索
R/次を検索	同じテキストをもう一度検索
C/置換...	テキストの置き換え
U/関数...	任意の関数の定義を検索
N/次のエラー	次のエラーを含む行を検索
P/前のエラー	前のエラーを含む行を検索

●M/メイク

C/コンパイルファイル	プログラムのコンパイル
B/プログラムファイル	コンパイルとリンク
R/再コンパイル	関連するすべてのプログラムをコンパイル／リンク
S/プログラムリストの設定...	プログラムに対する.MAKファイル設定
E/プログラムリストの編集...	プログラムリストの内容を変更
L/プログラムリストの削除	プログラムリストをメモリから削除

●R/実行

R/再実行	メモリにプログラムをロードし直す
G/実行	実行を開始
C/現在行まで実行	プログラムをカーソル位置まで実行
T/トレース	シングルステップトレース（関数の中もトレース）
S/ステップ	シングルステップトレース（関数の中はトレースしない）
A/アニメート	実行行を強調表示しながらプログラムを実行

●D/修正

C/関数...	呼び出している関数のリストを表示
B/ブレークポイント	ブレークポイントの設定／解除
P/ウオッチポイント	ウオッチポイントの設定／解除
W/ウオッチ変数	変数の値をモニタ
M/変数修正...	変数の値を修正
H/ヒストリ	デバッグ操作をファイルに記録
U/アンドゥ	ひとつ前のデバッグ操作に戻る
R/リプレイ	記録されたデバッグ操作を再現
T/インプット省略	デバッグ操作を再現するときに，保存されたユーザー入力を使わない

●U/補助メニュー

R/DOSコマンド実行	DOSコマンドを実行
C/自作メニュー	Utilityメニューに項目を追加
Q/QuickCの学習	QuickCの学習プログラムを起動

●O/環境

D/表示形式...	画面に表示される色や環境を変更
M/コンパイル・リンク...	コンパイラやリンカのオプション指定
R/実行・デバッグ...	実行およびデバッガのオプション指定
E/環境...	QuickCがファイルを検索するディレクトリの指定
F/フルメニュー	フルメニュー／ショートメニューの指定

●H/ヘルプ

I/索引	ヘルプの索引を表示
C/目次	ヘルプの目次を表示
T/キーワード	特定のトピックのヘルプを表示
H/ヘルプの利用法	ヘルプシステムの使い方を表示

■Borland Turbo C++ Version 1.01

●≡

About...	Turbo C++に関する情報の表示
Clear desktop	すべてのウィンドウをクローズし，ヒストリーリストをクリア
Repaint desktop	画面全体の再描画

●File

Open	既存ファイルのオープン
New	新規ファイルの作成
Save	編集中のファイルを保存
Save as...	編集中のファイルを名前を変えて保存
Save all	すべてのファイルを保存
Change dir...	ドライブまたはディレクトリの変更
Print	ファイルの印刷
Get info...	カレントファイルに関する情報を表示
DOS shell	DOSコマンドの実行
Quit	TurboC++の終了

●Edit

Restore line	最後の編集操作を取り消す
Cut	選択したテキストをクリップボードに移動
Copy	選択したテキストをクリップボードに複写
Paste	クリップボードのテキストを挿入
Copy example	ヘルプウィンドウの内容をクリップボードに複写
Show clipboard	クリップボードウィンドウのオープン
Clear	指定したテキストの削除

●Search

Find...	テキストの検索
Replace...	テキストの置き換え
Search again...	同じテキストをもう一度検索
Go to line number...	行番号による検索
Previous error	前のエラーを含む行の検索
Next error	次のエラーを含む行の検索
Locate function...	任意の関数の定義を検索

●Run

Run	プログラムの実行
Program reset	メモリにプログラムを再ロード
Go to cursor	プログラムを実行バーの位置からカーソル位置まで実行
Trace into	シングルステップトレース（関数の中もトレース）
Step over	シングルステップトレース（関数の中はトレースしない）
Arguments...	実行するプログラムに与える引数を設定

●Compile

Compile to OBJ	プログラムをコンパイルし，オブジェクトファイルを生成
Make EXE file	プロジェクトマネージャーを呼び，実行可能ファイルをメイク
Link EXE file	オブジェクトファイルをリンクして実行可能ファイルを生成
Build all	プロジェクト内のすべてのファイルを作り直す
Remove messages	メッセージウィンドウからすべてのメッセージを消去

●Debug

Inspect...	変数などの値を調べるためにInspectorウィンドウをオープン
Evaluate/modify...	変数または式を評価して，値の表示
Call stack...	呼び出しスタックの表示
Watches	ウォッチポイントの制御を設定
Toggle breakpoint	カーソル位置へのブレークポイントの設定あるいは解除
Breakpoints...	ブレークポイントの設定

●Project

Open project...	プロジェクトを選択し，オープン
Close project	プロジェクトをクローズし，デフォルトプロジェクトに戻る
Add item...	プロジェクトリストにファイルを追加
Delete item	プロジェクトリストからファイルを削除
Local options...	プロジェクトファイルの特定のモジュールのコマンドラインにオプションを追加
Include files...	インクルードするファイルの一覧を表示

●Option

Full menus	メニュー表示形式の設定
Compiler	コンパイラオプションの設定
Transfer...	≡メニューへ項目の追加
Make...	プロジェクト管理のための条件を設定
Linker...	リンクオプションの設定
Debugger...	組み込みデバッガに関するオプションの設定
Directries...	各ファイルのディレクトリ設定
Environment	使用環境の設定
Save...	環境・デスクトップ・プロジェクト関連の設定をファイルに保存

●Window

Size/Move	アクティブウィンドウのサイズまたは位置の変更
Zoom	アクティブウィンドウを最大または元のサイズに戻す
Tile	開いているすべてのウィンドウをタイリング
Cascade	開いているすべてのウィンドウをオーバーラップ
Next	次のウィンドウをアクティブに
Close	アクティブウィンドウをクローズ
Message	Messageウィンドウのオープン
Output	Outputウィンドウのオープン
Watch	Watchウィンドウのオープン
User screen	プログラム出力の表示
Register	Registerウィンドウのオープン
Project	Projectウィンドウのオープン
Project notes	プロジェクトファイルに関する情報などを書き込む
List...	すべての開いているウィンドウのリストを表示

●Help

Contents	ヘルプの目次を表示
Index	ヘルプの索引を表示
Topic search	特定のトピックに関するヘルプの表示
Previous topic	最後に表示したヘルプの表示
Help on help	ヘルプシステムの使い方を表示

■Borland Turbo Debugger Version 2.01

●≡

Repaint desktop	画面全体の再描画
Restore desktop	標準のウィンドウレイアウトのリストア
About...	Turbo Debuggerに関する情報の表示

●File

Open	デバッグプログラムのオープン
Change dir...	ドライブまたはディレクトリの変更
Get info...	プログラム情報を表示
DOS shell	DOSコマンドの実行
Resident	Turbo Debuggerを終了させてDOSに戻る
Symbol load...	.EXEファイルと関係なくシンボルテーブルのロード
Table relocate...	シンボルテーブルのベースセグメントの設定
Quit	Turbo Debuggerの終了

●View

Breakpoints...	ブレークポイントの表示
Stack	関数呼びだしスタックの表示
Log	イベントとデータのログの表示
Watches	ウォッチしている変数の表示
Variable	グローバル変数とローカル変数の表示
Module...	プログラムのソースモジュールの表示
File...	ファイルをアスキー形式または16進バイト形式で表示
CPU	CPUの命令・データ・スタックの表示
Dump	未処理のデータダンプの表示
Registers	CPUのレジスタとフラグの表示
Numeric processor	コプロセッサまたはエミュレータの表示
Exection history	バックトレースまたはキーストロークの再生のためにセーブされたアセンブラコードの表示
Hierarchy	オブジェクトまたはクラスの型リストと階層ツリーを表示
Another→	
＊Module	もうひとつのModuleウィンドウをオープン
＊Dump	もうひとつのDumpウィンドウをオープン
＊File...	もうひとつのFileウィンドウをオープン
Interrupt	Interruptウィンドウの表示

●Run

Run	プログラムを停止せずに実行
Go to cursor	プログラムをカーソル位置まで実行
Trace into	シングルステップトレース（関数の中もトレース）
Step over	シングルステップトレース（関数の中はトレースしない）
Execute to...	指定されたアドレスまで実行
Until return	関数から戻るまで実行
Animate...	プログラムを連続的にステップ実行
Back trace	プログラムをバックトレース
Instruction trace	1命令実行
Arguments...	プログラムに与える引数を設定
Program reset	現在のプログラムを再ロード

●Breakpoints

Toggle	カーソル位置でブレークポイントを設定・解除
At...	指定されたアドレスにブレークポイントを設定
Changed memory global...	メモリ領域にグローバルブレークポイントを設定
Expression true global...	式にグローバルブレークポイントを設定
Hardware breakpoint...	ハードウェアブレークポイントを設定
Delete all...	すべてのブレークポイントを解除

●Data

Inspect...	データオブジェクトを調べる
Evaluate/modify...	式を評価
Add watch...	Watchウィンドウに変数を追加
Function return	カレントルーチンの戻り値を調べる

●Option

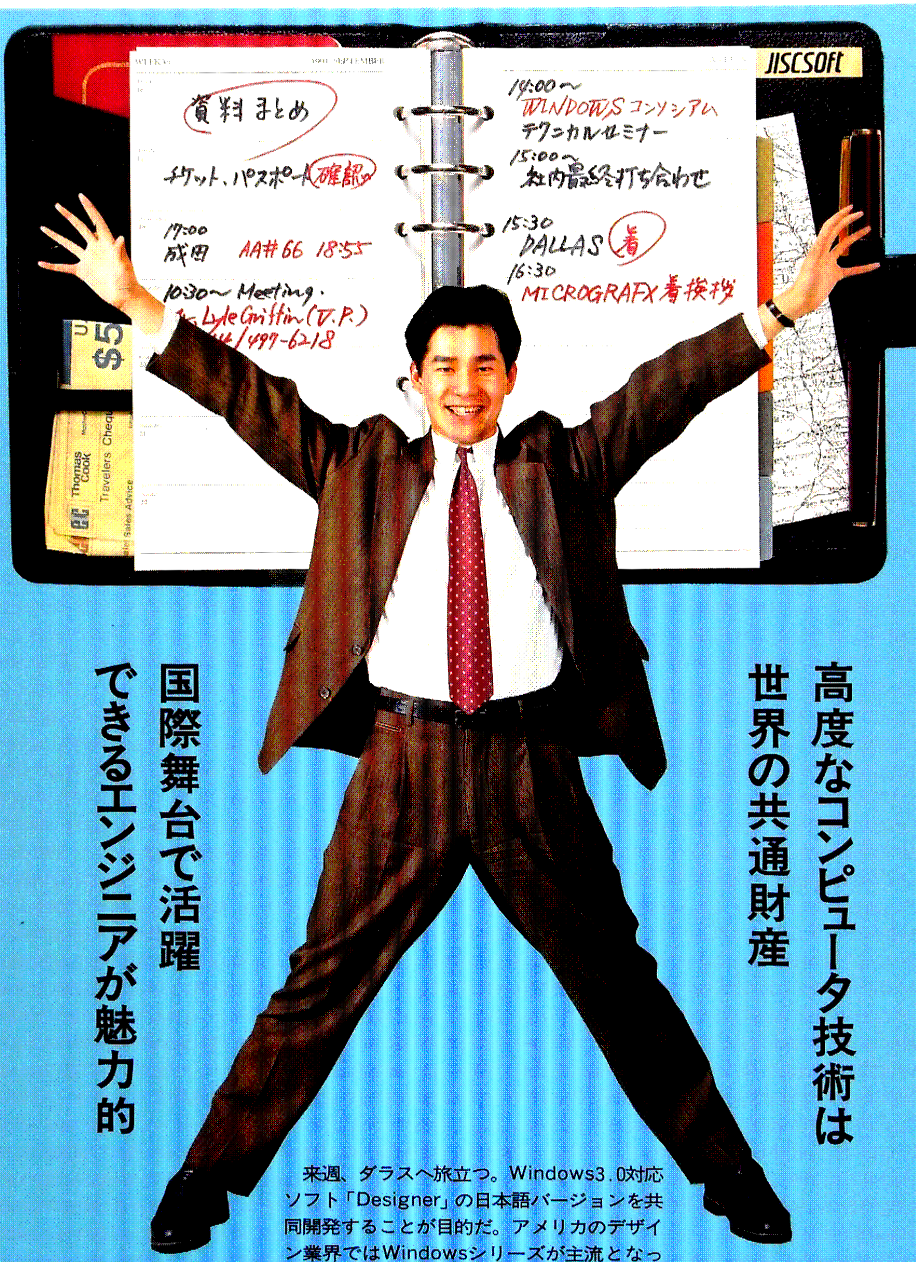
Language...	ソースモジュールから式の言語の設定
Macros→	
＊Create	キーストロークマクロの定義
＊Stop recording	記録セッションを終了
＊Remove	キーストロークマクロをひとつ削除
＊Delete all	すべてのキーストロークマクロを削除
Display options...	画面表示のオプションの設定
Path for source...	ソースファイルを探すディレクトリの設定
Save options...	オプション・マクロの内容を保存
Restore options...	オプションを読み込む
Hardware options...	グラフィックVRAMなどのシステムメモリへのアクセス方式や，カーソルの取り扱いなどハードウェア資源に関する設定

●Window

Zoom	アクティブウィンドウを最大または元のサイズに戻す
Next	次のウィンドウをアクティブに
Next pane	ウィンドウ内の次のペインに移動
Size/Move	アクティブウィンドウのサイズまたは位置の変更
Iconize/restore	カレントウィンドウをアイコン化，または元に戻す
Close	アクティブウィンドウを閉じる
Undo close	最後にクローズしたウィンドウを再びオープン
Dump pane to log	現在のペインをLogウィンドウに書き込む
User screen	プログラムの画面出力の表示

●Help

Index	ヘルプの索引を表示
Previous topic	最後に表示したヘルプの表示
Help on help	ヘルプシステムの使い方を表示



高度なコンピュータ技術は世界の共通財産

国際舞台で活躍できるエンジニアが魅力的

来週、ダラスへ旅立つ。Windows3.0対応ソフト「Designer」の日本語バージョンを共同開発することが目的だ。アメリカのデザイン業界ではWindowsシリーズが主流となっているが、まだまだ日本には普及していない。日本語対応パッケージの開発を急がなくては。マイクログラフィック社の副社長　Mr.Lyleに打ち合わせ時間は十分にとって頂こう。

しかし、出張目的はそれだけではない。アメリカの最先端コンピュータ技術と、それを使いこなす技術者の職場環境を体感することも目的のひとつ。日本という枠組みを超えたインターナショナルなソフトハウスだから、コンピュータの技術面においても、エンジニアを取り巻く開発環境においても、日本の現状を大きく進化させる可能性をJISCは秘めていると思う。ソフトウェア業界におけるボーダーレス時代も間近い。

ジスクソフト株式会社

●募集要項

職種／①システムエンジニア　②プログラマ　③グラフィックデザイナー

資格／専門学校卒以上　20歳〜30歳位迄の方
※経験者優遇（特に①②はEDP、DTP、UNIX、アセンブラ、C言語、COBOL、データベース、通信、ファームウェア、プロダクトソフト開発の経験者、優遇）
※未経験者歓迎（3年間じっくり勉強しながら、手がけたい開発分野を見つけてください。一人ひとりの個性を大切にし、丁寧な教育を実施）

給与／経験・年齢を考慮の上、当社規定により優遇します。
20歳　16万4750円（基本給15万2750円＋諸手当）　30歳SE例／27万8290円（交通費・残業代含まず）

待遇／昇給年1回、賞与年2回（2年度実績5ヵ月）、通勤交通費全額支給、各種社会保険完備、手当（職能・家族・残業・住宅・役職）、制度（退職金・財形貯蓄・住宅資金融資・社内旅行・クラブ（法人会員加入）

勤務地／本社（渋谷）、支社（豊橋）
※勤務地希望を考慮します。
※Uターン大歓迎します。
※海外勤務希望に応じます。
※海外出張もあります。

勤務時間／フレックスタイム制
※コアタイム10:00〜17:30
※全職種の実働は7.5時間です。

休日休暇／完全週休2日制（土・日）、祝日、慶弔、年末年始、夏季、特別休暇　※年間休日125日＋有給休暇（初年度10日、次年度12日、以降1年ごとに1日加算、最高30日）

応募／事前に電話連絡の上、履歴書（写真貼付）・職務経歴書（書式自由）をご持参ください。郵送も可。※今すぐ転職できない方の為の人材登録制度有
※応募の秘密厳守します。

事業内容／各種コンピュータの基本ソフトウェアの開発・製造・販売

交通／JR・東横・井の頭・新玉川・地下鉄半蔵門・銀座各線渋谷駅徒歩8分、地下鉄千代田・半蔵門・銀座各線表参道駅下車徒歩6分

●会社概要

設立／昭和45年6月

資本金／2400万円

売上高／10億7000万円（2年度実績）

社員数／60名（男：45名、女：15名）

関連会社／マイクログラフィックス・ジャパン、ジィスクソフトウェア・アメリカ（本社：テキサス州ダラス）他全12社

開発商品／Designer,
Charisma, Xport,
Voice Manager　他

主要取引先／富士通㈱、日本電気㈱、㈱アスキー、㈱日本ソフトバンク、㈱ソフトウェア・ジャパン、マイクログラフィックス他

所在地／〒150 東京都渋谷区渋谷2-2-10　青山H＆Aビル
Tel(03)3400-3121
担当：武車（ムシャ）、野田

●HOT LINE
フリーダイヤル　0120-275900

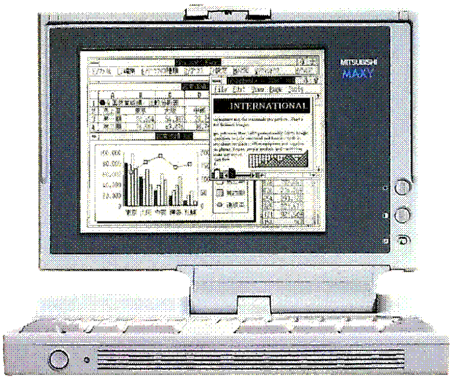
MITSUBISHI

技術がつくる高度なふれあい SOCIO-TECH



片手に乗った、スーパービジネス。

ライバルに差をつける、32ビットノートパソコン。



スーパービジネスは今、MAXYNOTE386から始まる。

「時代の最先端を切り開くビジネスエリートのために。」スーパービジネスに対応する、MAXYNOTE386の誕生です。A4ファイルサイズのコンパクトボディに、高速32ビットCPU386™SX(16MHz)を搭載。



さらにノート初の高速演算プロセッサ387™SXも実装可能(オプション)。広範なビジネスの高速処理を実現します。

着脱式大容量ハードディスクパックで、思いのままに、スーパービジネス。

片手でポンッと出し入れ自在、フロッピー感覚の超小形2.5インチハードディスクパック(40MB/20MB)を内蔵。大容量情報を効率的に、思い通りにコントロールできます。さらに、統合操作環境MS-Windows、統合ソフトMAXYMATEなどを標準装備。ビジネスパワーを高める即戦力ツールとして即座に活用できます。



MAXYSTATIONⅡで、さらに拡がる、スーパービジネス。

本体一体形のコンパクトな拡張ユニットMAXYSTATIONⅡをオプションで用意。PC/ATフルサイズカードが2枚利用可能です。また、100MBハードディスクも内蔵可能。PC/ATの豊富なハードウェア資産を活用した、多彩なネットワーク構築など、デスクトップに迫る豊かな拡張力を提供します。



●高性能CPU386™SX(16MHz)搭載、387™SX(オプション)●RAM標準1MB(最大5MB)●FDD 3.5インチ×1●HDD 2.5インチ、交換自在なハードディスクパック方式(40MB/20MB)●非同期内蔵モデム実装可(オプション)●RS-232C×1(最大2)●PC/ATフルサイズカード2枚実装可(オプション)●パワーONで瞬時に画面を再現するレジューム機能●MS-DOS、MS-Windows、MAXYMATE、MAXYメニューシステムを標準装備

AX do Next

世界標準機・三菱AXパソコン
MAXYNOTE386

標準価格●A12(HDD 20MBモデル)478,000円(税別)●A14(HDD 40MBモデル)578,000円(税別)〈この他に省スペースデスクトップタイプもあります〉

ニーズから発想する三菱情報通信システム Think&Link

AX THE ARCHITECTURE EXTENDED・AX

世界標準化を推進するAX。日本のソフトウェアメーカ、ハードウェアメーカ、流通各社によるAX協議会★のもと、OS、キーボード、画面、プリンタ部を規格化。国内・海外の豊富なソフトや周辺機器の共通利用を実現します。★1990年12月現在参加553社

三菱電機株式会社

お問合せは…三菱電機株式会社 〒100東京都千代田区丸の内2-2-3(三菱電機ビル) 本社情報処理機器事業部計画部(03)3218-3285 本社情報処理機器部(03)3218-3515 本社システム営業部(03)3218-3544 本社情報通信システム計画部(03)3218-3236 本社電機事業部(03)3218-3158 ●札幌(011)212-3739 ●仙台(022)264-5656 ●大宮(048)653-0261 ●横浜(045)212-3521 ●新潟(025)241-7220 ●金沢(0762)52-8821 ●名古屋(052)565-3246 ●大阪(06)347-2421 ●広島(082)248-5330 ●高松(0878)25-0009 ●福岡(092)721-2346

カタログ請求券 AXパソコン プログラマーズ 9102 あなたの電話番号をご記入ください

T4911777706988

〈資料請求番号P04〉

雑誌コード17777-06